Package ‘spatstat’

January 29, 2018

Version 1.55-0

Date 2018-01-29

Title Spatial Point Pattern Analysis, Model-Fitting, Simulation, Tests

Author Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>,
Rolf Turner <r.turner@auckland.ac.nz>
and Ege Rubak <rubak@math.aau.dk>,
with substantial contributions of code by
Kasper Klitgaard Berthelsen;
Ottmar Cronie;
Yongtao Guan;
Ute Hahn;
Abdollah Jalilian;
Marie-Colette van Lieshout;
Greg McSwiggan;
Tuomas Rajala;
Suman Rakshit;
Dominic Schuhmacher;
Rasmus Waagepetersen;
and Hangsheng Wang.
Additional contributions
by M. Adepeju;
C. Anderson;
Q.W. Ang;
M. Austenfeld;
S. Azaele;
M. Baddeley;
C. Beale;
M. Bell;
R. Bernhardt;
T. Bendtsen;
A. Bevan;
B. Biggerstaff;
A. Bilgrau;
L. Bischof;
C. Biscio;
R. Bivand;
J.M. Blanco Moreno;
F. Bonneu;
J. Burgos;

S. Byers;
Y.M. Chang;
J.B. Chen;
I. Chernayavsky;
Y.C. Chin;
B. Christensen;
J.-F. Coeurjolly;
K. Colyvas;
R. Constantine;
R. Corria Ainslie;
R. Cotton;
M. de la Cruz;
P. Dalgaard;
M. D'Antuono;
S. Das;
T. Davies;
P.J. Diggle;
P. Donnelly;
I. Dryden;
S. Eglen;
A. El-Gabbas;
B. Fandohan;
O. Flores;
E.D. Ford;
P. Forbes;
S. Frank;
J. Franklin;
N. Funwi-Gabga;
O. Garcia;
A. Gault;
J. Geldmann;
M. Genton;
S. Ghalandarayeshi;
J. Gilbey;
J. Goldstick;
P. Grabarnik;
C. Graf;
U. Hahn;
A. Hardegen;
M.B. Hansen;
M. Hazelton;
J. Heikkinen;
M. Hering;
M. Herrmann;
P. Hewson;
K. Hingee;
K. Hornik;
P. Hunziker;
J. Hywood;
R. Ihaka;
C. Icos;
A. Jammalamadaka;

R. John-Chandran;
D. Johnson;
M. Khanmohammadi;
R. Klaver;
P. Kovesi;
L. Kozmian-Ledward;
M. Kuhn;
J. Laake;
F. Lavancier;
T. Lawrence;
R.A. Lamb;
J. Lee;
G.P. Leser;
H.T. Li;
G. Limitsios;
A. Lister;
B. Madin;
M. Maechler;
J. Marcus;
K. Marchikanti;
R. Mark;
J. Mateu;
P. McCullagh;
U. Mehlig;
F. Mestre;
S. Meyer;
X.C. Mi;
L. De Middeleer;
R.K. Milne;
E. Miranda;
J. Moller;
M. Moradi;
V. Morera Pujol;
E. Mudrak;
G.M. Nair;
N. Najari;
N. Nava;
L.S. Nielsen;
F. Nunes;
J.R. Nyengaard;
J. Oehlschlaegel;
T. Onkelinx;
S. O'Riordan;
E. Parilov;
J. Picka;
N. Picard;
M. Porter;
S. Protsiv;
A. Raftery;
S. Rakshit;
B. Ramage;
P. Ramon;

X. Raynaud;
N. Read;
M. Reiter;
I. Renner;
T.O. Richardson;
B.D. Ripley;
E. Rosenbaum;
B. Rowlingson;
J. Rudokas;
J. Rudge;
C. Ryan;
F. Safavimanesh;
A. Sarkka;
C. Schank;
K. Schladitz;
S. Schutte;
B.T. Scott;
O. Semboli;
F. Semecurbe;
V. Shcherbakov;
G.C. Shen;
P. Shi;
H.-J. Ship;
T.L. Silva;
I.-M. Sintorn;
Y. Song;
M. Spiess;
M. Stevenson;
K. Stucki;
M. Sumner;
P. Surovy;
B. Taylor;
T. Thorarinsdottir;
L. Torres;
B. Turlach;
T. Tvedebrink;
K. Ummer;
M. Uppala;
A. van Burgel;
T. Verbeke;
M. Vihtakari;
A. Villers;
F. Vinatier;
S. Voss;
S. Wagner;
H. Wang;
H. Wendrock;
J. Wild;
C. Witthoft;
S. Wong;
M. Woringer;
M.E. Zamboni

and
A. Zeileis.

Maintainer Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Depends R (>= 3.3.0), spatstat.data (>= 1.2-0), stats, graphics, grDevices, utils, methods, nlme, rpart

Imports spatstat.utils (>= 1.8-0), mgcv, Matrix, deldir (>= 0.0-21), abind, tensor, polyclip (>= 1.5-0), goftest

Suggests sm, maptools, gsl, locfit, spatial, rpanel, tkrplot, RandomFields (>= 3.1.24.1), RandomFieldsUtils(>= 0.3.3.1), fftwtools (>= 0.9-8)

Description Comprehensive open-source toolbox for analysing Spatial Point Patterns. Focused mainly on two-dimensional point patterns, including multitype/marked points, in any spatial region. Also supports three-dimensional point patterns, space-time point patterns in any number of dimensions, point patterns on a linear network, and patterns of other geometrical objects. Supports spatial covariate data such as pixel images.
Contains over 2000 functions for plotting spatial data, exploratory data analysis, model-fitting, simulation, spatial sampling, model diagnostics, and formal inference.
Data types include point patterns, line segment patterns, spatial windows, pixel images, tessellations, and linear networks.
Exploratory methods include quadrat counts, K-functions and their simulation envelopes, nearest neighbour distance and empty space statistics, Fry plots, pair correlation function, kernel smoothed intensity, relative risk estimation with cross-validated bandwidth selection, mark correlation functions, segregation indices, mark dependence diagnostics, and kernel estimates of covariate effects. Formal hypothesis tests of random pattern (chi-squared, Kolmogorov-Smirnov, Monte Carlo, Diggle-Cressie-Loosmore-Ford, Dao-Genton, two-stage Monte Carlo) and tests for covariate effects (Cox-Berman-Waller-Lawson, Kolmogorov-Smirnov, ANOVA) are also supported.
Parametric models can be fitted to point pattern data using the functions ppm(), kppm(), slrm(), dppm() similar to glm(). Types of models include Poisson, Gibbs and Cox point processes, Neyman-Scott cluster processes, and determinantal point processes. Models may involve dependence on covariates, inter-point interaction, cluster formation and dependence on marks. Models are fitted by maximum likelihood, logistic regression, minimum contrast, and composite likelihood methods.
A model can be fitted to a list of point patterns (replicated point pattern data) using the function mppm(). The model can include random effects and fixed effects depending on the experimental design, in addition to all the features listed above.
Fitted point process models can be simulated, automatically. Formal hypothesis tests of a fitted model are supported (likelihood ratio test, analysis of deviance, Monte Carlo tests) along with basic tools for model selection (stepwise(), AIC()). Tools for validating the fitted model include simulation envelopes, residuals, residual plots and Q-Q plots, leverage and influence diagnostics, partial residuals, and added variable plots.

License GPL (>= 2)

URL http://www.spatstat.org

LazyData true

NeedsCompilation yes

ByteCompile true

BugReports https://github.com/spatstat/spatstat/issues

R topics documented:

spatstat-package *The Spatstat Package*

Description

This is a summary of the features of **spatstat**, a package in R for the statistical analysis of spatial point patterns.

Details

spatstat is a package for the statistical analysis of spatial data. Its main focus is the analysis of spatial patterns of points in two-dimensional space. The points may carry auxiliary data ('marks'), and the spatial region in which the points were recorded may have arbitrary shape.

The package is designed to support a complete statistical analysis of spatial data. It supports

- creation, manipulation and plotting of point patterns;
- exploratory data analysis;
- spatial random sampling;
- simulation of point process models;
- parametric model-fitting;
- non-parametric smoothing and regression;
- formal inference (hypothesis tests, confidence intervals);
- model diagnostics.

Apart from two-dimensional point patterns and point processes, **spatstat** also supports point patterns in three dimensions, point patterns in multidimensional space-time, point patterns on a linear network, patterns of line segments in two dimensions, and spatial tessellations and random sets in two dimensions.

The package can fit several types of point process models to a point pattern dataset:

- Poisson point process models (by Berman-Turner approximate maximum likelihood or by spatial logistic regression)
- Gibbs/Markov point process models (by Baddeley-Turner approximate maximum pseudolikelihood, Coeurjolly-Rubak logistic likelihood, or Huang-Ogata approximate maximum likelihood)
- Cox/cluster point process models (by Waagepetersen's two-step fitting procedure and minimum contrast, composite likelihood, or Palm likelihood)
- determinantal point process models (by Waagepetersen's two-step fitting procedure and minimum contrast, composite likelihood, or Palm likelihood)

The models may include spatial trend, dependence on covariates, and complicated interpoint interactions. Models are specified by a `formula` in the R language, and are fitted using a function analogous to `lm` and `glm`. Fitted models can be printed, plotted, predicted, simulated and so on.

Getting Started

For a quick introduction to **spatstat**, read the package vignette *Getting started with spatstat* installed with **spatstat**. To read that document, you can either

- visit `cran.r-project.org/web/packages/spatstat` and click on `Getting Started with Spatstat`
- start R, type `library(spatstat)` and `vignette('getstart')`
- start R, type `help.start()` to open the help browser, and navigate to `Packages > spatstat > Vignettes`.

Once you have installed **spatstat**, start R and type `library(spatstat)`. Then type `beginner` for a beginner's introduction, or `demo(spatstat)` for a demonstration of the package's capabilities.

For a complete course on **spatstat**, and on statistical analysis of spatial point patterns, read the book by Baddeley, Rubak and Turner (2015). Other recommended books on spatial point process methods are Diggle (2014), Gelfand et al (2010) and Illian et al (2008).

The **spatstat** package includes over 50 datasets, which can be useful when learning the package. Type `demo(data)` to see plots of all datasets available in the package. Type `vignette('datasets')` for detailed background information on these datasets, and plots of each dataset.

For information on converting your data into **spatstat** format, read Chapter 3 of Baddeley, Rubak and Turner (2015). This chapter is available free online, as one of the sample chapters at the book companion website, `spatstat.github.io/book`.

For information about handling data in **shapefiles**, see Chapter 3, or the Vignette *Handling shapefiles in the spatstat package*, installed with **spatstat**, accessible as `vignette('shapefiles')`.

Updates

New versions of **spatstat** are released every 8 weeks. Users are advised to update their installation of **spatstat** regularly.

Type `latest.news` to read the news documentation about changes to the current installed version of **spatstat**.

See the Vignette *Summary of recent updates*, installed with **spatstat**, which describes the main changes to **spatstat** since the book (Baddeley, Rubak and Turner, 2015) was published. It is accessible as `vignette('updates')`.

Type `news(package="spatstat")` to read news documentation about all previous versions of the package.

FUNCTIONS AND DATASETS

Following is a summary of the main functions and datasets in the **spatstat** package. Alternatively an alphabetical list of all functions and datasets is available by typing `library(help=spatstat)`.

For further information on any of these, type `help(name)` or `?name` where `name` is the name of the function or dataset.

CONTENTS:

I. CREATING AND MANIPULATING DATA

Types of spatial data:

The main types of spatial data supported by **spatstat** are:

ppp	point pattern
owin	window (spatial region)
im	pixel image
psp	line segment pattern
tess	tessellation
pp3	three-dimensional point pattern
ppx	point pattern in any number of dimensions
lpp	point pattern on a linear network

To create a point pattern:

ppp	create a point pattern from (x, y) and window information `ppp(x, y, xlim, ylim)` for rectangular window `ppp(x, y, poly)` for polygonal window `ppp(x, y, mask)` for binary image window
as.ppp	convert other types of data to a ppp object
clickppp	interactively add points to a plot
marks<-, %mark%	attach/reassign marks to a point pattern

To simulate a random point pattern:

runifpoint	generate n independent uniform random points
rpoint	generate n independent random points
rmpoint	generate n independent multitype random points
rpoispp	simulate the (in)homogeneous Poisson point process
rmpoispp	simulate the (in)homogeneous multitype Poisson point process
runifdisc	generate n independent uniform random points in disc
rstrat	stratified random sample of points
rsyst	systematic random sample of points
rjitter	apply random displacements to points in a pattern
rMaternI	simulate the Matérn Model I inhibition process
rMaternII	simulate the Matérn Model II inhibition process
rSSI	simulate Simple Sequential Inhibition process
rStrauss	simulate Strauss process (perfect simulation)
rHardcore	simulate Hard Core process (perfect simulation)
rStraussHard	simulate Strauss-hard core process (perfect simulation)
rDiggleGratton	simulate Diggle-Gratton process (perfect simulation)
rDGS	simulate Diggle-Gates-Stibbard process (perfect simulation)
rPenttinen	simulate Penttinen process (perfect simulation)
rNeymanScott	simulate a general Neyman-Scott process
rPoissonCluster	simulate a general Poisson cluster process
rMatClust	simulate the Matérn Cluster process

rThomas	simulate the Thomas process
rGaussPoisson	simulate the Gauss-Poisson cluster process
rCauchy	simulate Neyman-Scott Cauchy cluster process
rVarGamma	simulate Neyman-Scott Variance Gamma cluster process
rthin	random thinning
rcell	simulate the Baddeley-Silverman cell process
rmh	simulate Gibbs point process using Metropolis-Hastings
simulate.ppm	simulate Gibbs point process using Metropolis-Hastings
runifpointOnLines	generate n random points along specified line segments
rpoisppOnLines	generate Poisson random points along specified line segments

To randomly change an existing point pattern:

rshift	random shifting of points
rjitter	apply random displacements to points in a pattern
rthin	random thinning
rlabel	random (re)labelling of a multitype point pattern
quadratresample	block resampling

Standard point pattern datasets:

Datasets in **spatstat** are lazy-loaded, so you can simply type the name of the dataset to use it; there is no need to type `data(amacrine)` etc.

Type `demo(data)` to see a display of all the datasets installed with the package.

Type `vignette('datasets')` for a document giving an overview of all datasets, including background information, and plots.

amacrine	Austin Hughes' rabbit amacrine cells
anemones	Upton-Fingleton sea anemones data
ants	Harkness-Isham ant nests data
bdspots	Breakdown spots in microelectrodes
bei	Tropical rainforest trees
betacells	Waessle et al. cat retinal ganglia data
bramblecanes	Bramble Canes data
bronzefilter	Bronze Filter Section data
cells	Crick-Ripley biological cells data
chicago	Chicago crimes
chorley	Chorley-Ribble cancer data
clmfires	Castilla-La Mancha forest fires
copper	Berman-Huntington copper deposits data
dendrite	Dendritic spines
demohyper	Synthetic point patterns
demopat	Synthetic point pattern
finpines	Finnish Pines data
flu	Influenza virus proteins
gordon	People in Gordon Square, London
gorillas	Gorilla nest sites
hamster	Aherne's hamster tumour data
humberside	North Humberside childhood leukaemia data
hyytiala	Mixed forest in Hyytiälä, Finland
japanesepines	Japanese Pines data
lansing	Lansing Woods data

longleaf	Longleaf Pines data
mucosa	Cells in gastric mucosa
murchison	Murchison gold deposits
nbfires	New Brunswick fires data
nztrees	Mark-Esler-Ripley trees data
osteo	Osteocyte lacunae (3D, replicated)
paracou	Kimboto trees in Paracou, French Guiana
ponderosa	Getis-Franklin ponderosa pine trees data
pyramidal	Pyramidal neurons from 31 brains
redwood	Strauss-Ripley redwood saplings data
redwoodfull	Strauss redwood saplings data (full set)
residualspaper	Data from Baddeley et al (2005)
shapley	Galaxies in an astronomical survey
simdat	Simulated point pattern (inhomogeneous, with interaction)
spiders	Spider webs on mortar lines of brick wall
sporophores	Mycorrhizal fungi around a tree
spruces	Spruce trees in Saxonia
swedishpines	Strand-Ripley Swedish pines data
urkiola	Urkiola Woods data
waka	Trees in Waka national park
waterstriders	Insects on water surface

To manipulate a point pattern:

plot.ppp	plot a point pattern (e.g. plot(X))
iplot	plot a point pattern interactively
edit.ppp	interactive text editor
[.ppp	extract or replace a subset of a point pattern pp[subset] or pp[subwindow]
subset.ppp	extract subset of point pattern satisfying a condition
superimpose	combine several point patterns
by.ppp	apply a function to sub-patterns of a point pattern
cut.ppp	classify the points in a point pattern
split.ppp	divide pattern into sub-patterns
unmark	remove marks
npoints	count the number of points
coords	extract coordinates, change coordinates
marks	extract marks, change marks or attach marks
rotate	rotate pattern
shift	translate pattern
flipxy	swap x and y coordinates
reflect	reflect in the origin
periodify	make several translated copies
affine	apply affine transformation
scalardilate	apply scalar dilation
density.ppp	kernel estimation of point pattern intensity
Smooth.ppp	kernel smoothing of marks of point pattern
nnmark	mark value of nearest data point
sharpen.ppp	data sharpening
identify.ppp	interactively identify points
unique.ppp	remove duplicate points
duplicated.ppp	determine which points are duplicates
connected.ppp	find clumps of points

dirichlet	compute Dirichlet-Voronoi tessellation
delaunay	compute Delaunay triangulation
delaunayDistance	graph distance in Delaunay triangulation
convexhull	compute convex hull
discretise	discretise coordinates
pixellate.ppp	approximate point pattern by pixel image
as.im.ppp	approximate point pattern by pixel image

See `spatstat.options` to control plotting behaviour.

To create a window:

An object of class `"owin"` describes a spatial region (a window of observation).

owin	Create a window object `owin(xlim, ylim)` for rectangular window `owin(poly)` for polygonal window `owin(mask)` for binary image window
Window	Extract window of another object
Frame	Extract the containing rectangle ('frame') of another object
as.owin	Convert other data to a window object
square	make a square window
disc	make a circular window
ellipse	make an elliptical window
ripras	Ripley-Rasson estimator of window, given only the points
convexhull	compute convex hull of something
letterR	polygonal window in the shape of the R logo
clickpoly	interactively draw a polygonal window
clickbox	interactively draw a rectangle

To manipulate a window:

plot.owin	plot a window. `plot(W)`
boundingbox	Find a tight bounding box for the window
erosion	erode window by a distance r
dilation	dilate window by a distance r
closing	close window by a distance r
opening	open window by a distance r
border	difference between window and its erosion/dilation
complement.owin	invert (swap inside and outside)
simplify.owin	approximate a window by a simple polygon
rotate	rotate window
flipxy	swap x and y coordinates
shift	translate window
periodify	make several translated copies
affine	apply affine transformation
as.data.frame.owin	convert window to data frame

Digital approximations:

as.mask	Make a discrete pixel approximation of a given window
as.im.owin	convert window to pixel image

pixellate.owin	convert window to pixel image
commonGrid	find common pixel grid for windows
nearest.raster.point	map continuous coordinates to raster locations
raster.x	raster x coordinates
raster.y	raster y coordinates
raster.xy	raster x and y coordinates
as.polygonal	convert pixel mask to polygonal window

See spatstat.options to control the approximation

Geometrical computations with windows:

edges	extract boundary edges
intersect.owin	intersection of two windows
union.owin	union of two windows
setminus.owin	set subtraction of two windows
inside.owin	determine whether a point is inside a window
area.owin	compute area
perimeter	compute perimeter length
diameter.owin	compute diameter
incircle	find largest circle inside a window
inradius	radius of incircle
connected.owin	find connected components of window
eroded.areas	compute areas of eroded windows
dilated.areas	compute areas of dilated windows
bdist.points	compute distances from data points to window boundary
bdist.pixels	compute distances from all pixels to window boundary
bdist.tiles	boundary distance for each tile in tessellation
distmap.owin	distance transform image
distfun.owin	distance transform
centroid.owin	compute centroid (centre of mass) of window
is.subset.owin	determine whether one window contains another
is.convex	determine whether a window is convex
convexhull	compute convex hull
triangulate.owin	decompose into triangles
as.mask	pixel approximation of window
as.polygonal	polygonal approximation of window
is.rectangle	test whether window is a rectangle
is.polygonal	test whether window is polygonal
is.mask	test whether window is a mask
setcov	spatial covariance function of window
pixelcentres	extract centres of pixels in mask
clickdist	measure distance between two points clicked by user

Pixel images: An object of class "im" represents a pixel image. Such objects are returned by some of the functions in **spatstat** including Kmeasure, setcov and density.ppp.

im	create a pixel image
as.im	convert other data to a pixel image
pixellate	convert other data to a pixel image
as.matrix.im	convert pixel image to matrix
as.data.frame.im	convert pixel image to data frame
as.function.im	convert pixel image to function

plot.im	plot a pixel image on screen as a digital image
contour.im	draw contours of a pixel image
persp.im	draw perspective plot of a pixel image
rgbim	create colour-valued pixel image
hsvim	create colour-valued pixel image
[.im	extract a subset of a pixel image
[<-.im	replace a subset of a pixel image
rotate.im	rotate pixel image
shift.im	apply vector shift to pixel image
affine.im	apply affine transformation to image
X	print very basic information about image X
summary(X)	summary of image X
hist.im	histogram of image
mean.im	mean pixel value of image
integral.im	integral of pixel values
quantile.im	quantiles of image
cut.im	convert numeric image to factor image
is.im	test whether an object is a pixel image
interp.im	interpolate a pixel image
blur	apply Gaussian blur to image
Smooth.im	apply Gaussian blur to image
connected.im	find connected components
compatible.im	test whether two images have compatible dimensions
harmonise.im	make images compatible
commonGrid	find a common pixel grid for images
eval.im	evaluate any expression involving images
scaletointerval	rescale pixel values
zapsmall.im	set very small pixel values to zero
levelset	level set of an image
solutionset	region where an expression is true
imcov	spatial covariance function of image
convolve.im	spatial convolution of images
transect.im	line transect of image
pixelcentres	extract centres of pixels
transmat	convert matrix of pixel values
	to a different indexing convention
rnoise	random pixel noise

Line segment patterns

An object of class "psp" represents a pattern of straight line segments.

psp	create a line segment pattern
as.psp	convert other data into a line segment pattern
edges	extract edges of a window
is.psp	determine whether a dataset has class "psp"
plot.psp	plot a line segment pattern
print.psp	print basic information
summary.psp	print summary information
[.psp	extract a subset of a line segment pattern
as.data.frame.psp	convert line segment pattern to data frame
marks.psp	extract marks of line segments
marks<-.psp	assign new marks to line segments

unmark.psp	delete marks from line segments
midpoints.psp	compute the midpoints of line segments
endpoints.psp	extract the endpoints of line segments
lengths.psp	compute the lengths of line segments
angles.psp	compute the orientation angles of line segments
superimpose	combine several line segment patterns
flipxy	swap x and y coordinates
rotate.psp	rotate a line segment pattern
shift.psp	shift a line segment pattern
periodify	make several shifted copies
affine.psp	apply an affine transformation
pixellate.psp	approximate line segment pattern by pixel image
as.mask.psp	approximate line segment pattern by binary mask
distmap.psp	compute the distance map of a line segment pattern
distfun.psp	compute the distance map of a line segment pattern
density.psp	kernel smoothing of line segments
selfcrossing.psp	find crossing points between line segments
selfcut.psp	cut segments where they cross
crossing.psp	find crossing points between two line segment patterns
nncross	find distance to nearest line segment from a given point
nearestsegment	find line segment closest to a given point
project2segment	find location along a line segment closest to a given point
pointsOnLines	generate points evenly spaced along line segment
rpoisline	generate a realisation of the Poisson line process inside a window
rlinegrid	generate a random array of parallel lines through a window

Tessellations

An object of class "tess" represents a tessellation.

tess	create a tessellation
quadrats	create a tessellation of rectangles
hextess	create a tessellation of hexagons
quantess	quantile tessellation
as.tess	convert other data to a tessellation
plot.tess	plot a tessellation
tiles	extract all the tiles of a tessellation
[.tess	extract some tiles of a tessellation
[<-.tess	change some tiles of a tessellation
intersect.tess	intersect two tessellations
	or restrict a tessellation to a window
chop.tess	subdivide a tessellation by a line
dirichlet	compute Dirichlet-Voronoi tessellation of points
delaunay	compute Delaunay triangulation of points
rpoislinetess	generate tessellation using Poisson line process
tile.areas	area of each tile in tessellation
bdist.tiles	boundary distance for each tile in tessellation

Three-dimensional point patterns

An object of class "pp3" represents a three-dimensional point pattern in a rectangular box. The box is represented by an object of class "box3".

pp3	create a 3-D point pattern

plot.pp3	plot a 3-D point pattern
coords	extract coordinates
as.hyperframe	extract coordinates
subset.pp3	extract subset of 3-D point pattern
unitname.pp3	name of unit of length
npoints	count the number of points
runifpoint3	generate uniform random points in 3-D
rpoispp3	generate Poisson random points in 3-D
envelope.pp3	generate simulation envelopes for 3-D pattern
box3	create a 3-D rectangular box
as.box3	convert data to 3-D rectangular box
unitname.box3	name of unit of length
diameter.box3	diameter of box
volume.box3	volume of box
shortside.box3	shortest side of box
eroded.volumes	volumes of erosions of box

Multi-dimensional space-time point patterns

An object of class "ppx" represents a point pattern in multi-dimensional space and/or time.

ppx	create a multidimensional space-time point pattern
coords	extract coordinates
as.hyperframe	extract coordinates
subset.ppx	extract subset
unitname.ppx	name of unit of length
npoints	count the number of points
runifpointx	generate uniform random points
rpoisppx	generate Poisson random points
boxx	define multidimensional box
diameter.boxx	diameter of box
volume.boxx	volume of box
shortside.boxx	shortest side of box
eroded.volumes.boxx	volumes of erosions of box

Point patterns on a linear network

An object of class "linnet" represents a linear network (for example, a road network).

linnet	create a linear network
clickjoin	interactively join vertices in network
iplot.linnet	interactively plot network
simplenet	simple example of network
lineardisc	disc in a linear network
delaunayNetwork	network of Delaunay triangulation
dirichletNetwork	network of Dirichlet edges
methods.linnet	methods for linnet objects
vertices.linnet	nodes of network
pixellate.linnet	approximate by pixel image

An object of class "lpp" represents a point pattern on a linear network (for example, road accidents on a road network).

lpp	create a point pattern on a linear network
methods.lpp	methods for lpp objects
subset.lpp	method for subset
rpoislpp	simulate Poisson points on linear network
runiflpp	simulate random points on a linear network
chicago	Chicago crime data
dendrite	Dendritic spines data
spiders	Spider webs on mortar lines of brick wall

Hyperframes

A hyperframe is like a data frame, except that the entries may be objects of any kind.

hyperframe	create a hyperframe
as.hyperframe	convert data to hyperframe
plot.hyperframe	plot hyperframe
with.hyperframe	evaluate expression using each row of hyperframe
cbind.hyperframe	combine hyperframes by columns
rbind.hyperframe	combine hyperframes by rows
as.data.frame.hyperframe	convert hyperframe to data frame
subset.hyperframe	method for subset
head.hyperframe	first few rows of hyperframe
tail.hyperframe	last few rows of hyperframe

Layered objects

A layered object represents data that should be plotted in successive layers, for example, a background and a foreground.

layered	create layered object
plot.layered	plot layered object
[.layered	extract subset of layered object

Colour maps

A colour map is a mechanism for associating colours with data. It can be regarded as a function, mapping data to colours. Using a colourmap object in a plot command ensures that the mapping from numbers to colours is the same in different plots.

colourmap	create a colour map
plot.colourmap	plot the colour map only
tweak.colourmap	alter individual colour values
interp.colourmap	make a smooth transition between colours
beachcolourmap	one special colour map

II. EXPLORATORY DATA ANALYSIS

Inspection of data:

summary(X)	print useful summary of point pattern X
X	print basic description of point pattern X
any(duplicated(X))	check for duplicated points in pattern X
istat(X)	Interactive exploratory analysis
View(X)	spreadsheet-style viewer

Classical exploratory tools:

clarkevans	Clark and Evans aggregation index
fryplot	Fry plot
miplot	Morisita Index plot

Smoothing:

density.ppp	kernel smoothed density/intensity
relrisk	kernel estimate of relative risk
Smooth.ppp	spatial interpolation of marks
bw.diggle	cross-validated bandwidth selection for density.ppp
bw.ppl	likelihood cross-validated bandwidth selection for density.ppp
bw.scott	Scott's rule of thumb for density estimation
bw.relrisk	cross-validated bandwidth selection for relrisk
bw.smoothppp	cross-validated bandwidth selection for Smooth.ppp
bw.frac	bandwidth selection using window geometry
bw.stoyan	Stoyan's rule of thumb for bandwidth for pcf

Modern exploratory tools:

clusterset	Allard-Fraley feature detection
nnclean	Byers-Raftery feature detection
sharpen.ppp	Choi-Hall data sharpening
rhohat	Kernel estimate of covariate effect
rho2hat	Kernel estimate of effect of two covariates
spatialcdf	Spatial cumulative distribution function
roc	Receiver operating characteristic curve

Summary statistics for a point pattern: Type `demo(sumfun)` for a demonstration of many of the summary statistics.

intensity	Mean intensity
quadratcount	Quadrat counts
intensity.quadratcount	Mean intensity in quadrats
Fest	empty space function F
Gest	nearest neighbour distribution function G
Jest	J-function $J = (1 - G)/(1 - F)$
Kest	Ripley's K-function
Lest	Besag L-function
Tstat	Third order T-function
allstats	all four functions F, G, J, K
pcf	pair correlation function
Kinhom	K for inhomogeneous point patterns
Linhom	L for inhomogeneous point patterns
pcfinhom	pair correlation for inhomogeneous patterns
Finhom	F for inhomogeneous point patterns
Ginhom	G for inhomogeneous point patterns
Jinhom	J for inhomogeneous point patterns
localL	Getis-Franklin neighbourhood density function
localK	neighbourhood K-function
localpcf	local pair correlation function

`localKinhom`	local K for inhomogeneous point patterns
`localLinhom`	local L for inhomogeneous point patterns
`localpcfinhom`	local pair correlation for inhomogeneous patterns
`Ksector`	Directional K-function
`Kscaled`	locally scaled K-function
`Kest.fft`	fast K-function using FFT for large datasets
`Kmeasure`	reduced second moment measure
`envelope`	simulation envelopes for a summary function
`varblock`	variances and confidence intervals for a summary function
`lohboot`	bootstrap for a summary function

Related facilities:

`plot.fv`	plot a summary function
`eval.fv`	evaluate any expression involving summary functions
`harmonise.fv`	make functions compatible
`eval.fasp`	evaluate any expression involving an array of functions
`with.fv`	evaluate an expression for a summary function
`Smooth.fv`	apply smoothing to a summary function
`deriv.fv`	calculate derivative of a summary function
`pool.fv`	pool several estimates of a summary function
`nndist`	nearest neighbour distances
`nnwhich`	find nearest neighbours
`pairdist`	distances between all pairs of points
`crossdist`	distances between points in two patterns
`nncross`	nearest neighbours between two point patterns
`exactdt`	distance from any location to nearest data point
`distmap`	distance map image
`distfun`	distance map function
`nnmap`	nearest point image
`nnfun`	nearest point function
`density.ppp`	kernel smoothed density
`Smooth.ppp`	spatial interpolation of marks
`relrisk`	kernel estimate of relative risk
`sharpen.ppp`	data sharpening
`rknn`	theoretical distribution of nearest neighbour distance

Summary statistics for a multitype point pattern: A multitype point pattern is represented by an object `X` of class `"ppp"` such that `marks(X)` is a factor.

`relrisk`	kernel estimation of relative risk
`scan.test`	spatial scan test of elevated risk
`Gcross`,`Gdot`,`Gmulti`	multitype nearest neighbour distributions $G_{ij}, G_{i\bullet}$
`Kcross`,`Kdot`, `Kmulti`	multitype K-functions $K_{ij}, K_{i\bullet}$
`Lcross`,`Ldot`	multitype L-functions $L_{ij}, L_{i\bullet}$
`Jcross`,`Jdot`,`Jmulti`	multitype J-functions $J_{ij}, J_{i\bullet}$
`pcfcross`	multitype pair correlation function g_{ij}
`pcfdot`	multitype pair correlation function $g_{i\bullet}$
`pcfmulti`	general pair correlation function
`markconnect`	marked connection function p_{ij}
`alltypes`	estimates of the above for all i, j pairs

Iest	multitype I-function
Kcross.inhom,Kdot.inhom	inhomogeneous counterparts of Kcross, Kdot
Lcross.inhom,Ldot.inhom	inhomogeneous counterparts of Lcross, Ldot
pcfcross.inhom,pcfdot.inhom	inhomogeneous counterparts of pcfcross, pcfdot

Summary statistics for a marked point pattern: A marked point pattern is represented by an object X of class "ppp" with a component X$marks. The entries in the vector X$marks may be numeric, complex, string or any other atomic type. For numeric marks, there are the following functions:

markmean	smoothed local average of marks
markvar	smoothed local variance of marks
markcorr	mark correlation function
markcrosscorr	mark cross-correlation function
markvario	mark variogram
Kmark	mark-weighted K function
Emark	mark independence diagnostic $E(r)$
Vmark	mark independence diagnostic $V(r)$
nnmean	nearest neighbour mean index
nnvario	nearest neighbour mark variance index

For marks of any type, there are the following:

Gmulti	multitype nearest neighbour distribution
Kmulti	multitype K-function
Jmulti	multitype J-function

Alternatively use cut.ppp to convert a marked point pattern to a multitype point pattern.

Programming tools:

applynbd	apply function to every neighbourhood in a point pattern
markstat	apply function to the marks of neighbours in a point pattern
marktable	tabulate the marks of neighbours in a point pattern
pppdist	find the optimal match between two point patterns

Summary statistics for a point pattern on a linear network:

These are for point patterns on a linear network (class lpp). For unmarked patterns:

linearK	K function on linear network
linearKinhom	inhomogeneous K function on linear network
linearpcf	pair correlation function on linear network
linearpcfinhom	inhomogeneous pair correlation on linear network

For multitype patterns:

linearKcross	K function between two types of points
linearKdot	K function from one type to any type
linearKcross.inhom	Inhomogeneous version of linearKcross
linearKdot.inhom	Inhomogeneous version of linearKdot
linearmarkconnect	Mark connection function on linear network

linearmarkequal	Mark equality function on linear network
linearpcfcross	Pair correlation between two types of points
linearpcfdot	Pair correlation from one type to any type
linearpcfcross.inhom	Inhomogeneous version of linearpcfcross
linearpcfdot.inhom	Inhomogeneous version of linearpcfdot

Related facilities:

pairdist.lpp	distances between pairs
crossdist.lpp	distances between pairs
nndist.lpp	nearest neighbour distances
nncross.lpp	nearest neighbour distances
nnwhich.lpp	find nearest neighbours
nnfun.lpp	find nearest data point
density.lpp	kernel smoothing estimator of intensity
distfun.lpp	distance transform
envelope.lpp	simulation envelopes
rpoislpp	simulate Poisson points on linear network
runiflpp	simulate random points on a linear network

It is also possible to fit point process models to lpp objects. See Section IV.

Summary statistics for a three-dimensional point pattern:

These are for 3-dimensional point pattern objects (class pp3).

F3est	empty space function F
G3est	nearest neighbour function G
K3est	K-function
pcf3est	pair correlation function

Related facilities:

envelope.pp3	simulation envelopes
pairdist.pp3	distances between all pairs of points
crossdist.pp3	distances between points in two patterns
nndist.pp3	nearest neighbour distances
nnwhich.pp3	find nearest neighbours
nncross.pp3	find nearest neighbours in another pattern

Computations for multi-dimensional point pattern:

These are for multi-dimensional space-time point pattern objects (class ppx).

pairdist.ppx	distances between all pairs of points
crossdist.ppx	distances between points in two patterns
nndist.ppx	nearest neighbour distances
nnwhich.ppx	find nearest neighbours

Summary statistics for random sets:

These work for point patterns (class ppp), line segment patterns (class psp) or windows (class owin).

Hest	spherical contact distribution H

Gfox	Foxall G-function
Jfox	Foxall J-function

III. MODEL FITTING (COX AND CLUSTER MODELS)

Cluster process models (with homogeneous or inhomogeneous intensity) and Cox processes can be fitted by the function kppm. Its result is an object of class "kppm". The fitted model can be printed, plotted, predicted, simulated and updated.

kppm	Fit model
plot.kppm	Plot the fitted model
summary.kppm	Summarise the fitted model
fitted.kppm	Compute fitted intensity
predict.kppm	Compute fitted intensity
update.kppm	Update the model
improve.kppm	Refine the estimate of trend
simulate.kppm	Generate simulated realisations
vcov.kppm	Variance-covariance matrix of coefficients
coef.kppm	Extract trend coefficients
formula.kppm	Extract trend formula
parameters	Extract all model parameters
clusterfield	Compute offspring density
clusterradius	Radius of support of offspring density
Kmodel.kppm	K function of fitted model
pcfmodel.kppm	Pair correlation of fitted model

For model selection, you can also use the generic functions step, drop1 and AIC on fitted point process models.

The theoretical models can also be simulated, for any choice of parameter values, using rThomas, rMatClust, rCauchy, rVarGamma, and rLGCP.

Lower-level fitting functions include:

lgcp.estK	fit a log-Gaussian Cox process model
lgcp.estpcf	fit a log-Gaussian Cox process model
thomas.estK	fit the Thomas process model
thomas.estpcf	fit the Thomas process model
matclust.estK	fit the Matern Cluster process model
matclust.estpcf	fit the Matern Cluster process model
cauchy.estK	fit a Neyman-Scott Cauchy cluster process
cauchy.estpcf	fit a Neyman-Scott Cauchy cluster process
vargamma.estK	fit a Neyman-Scott Variance Gamma process
vargamma.estpcf	fit a Neyman-Scott Variance Gamma process
mincontrast	low-level algorithm for fitting models by the method of minimum contrast

IV. MODEL FITTING (POISSON AND GIBBS MODELS)

Types of models

Poisson point processes are the simplest models for point patterns. A Poisson model assumes that the points are stochastically independent. It may allow the points to have a non-uniform spatial density. The special case of a Poisson process with a uniform spatial density is often called Complete Spatial Randomness.

Poisson point processes are included in the more general class of Gibbs point process models. In a Gibbs model, there is *interaction* or dependence between points. Many different types of interaction can be specified.

For a detailed explanation of how to fit Poisson or Gibbs point process models to point pattern data using **spatstat**, see Baddeley and Turner (2005b) or Baddeley (2008).

To fit a Poisson or Gibbs point process model:

Model fitting in **spatstat** is performed mainly by the function `ppm`. Its result is an object of class `"ppm"`.

Here are some examples, where `X` is a point pattern (class `"ppp"`):

command	*model*
`ppm(X)`	Complete Spatial Randomness
`ppm(X ~ 1)`	Complete Spatial Randomness
`ppm(X ~ x)`	Poisson process with intensity loglinear in x coordinate
`ppm(X ~ 1, Strauss(0.1))`	Stationary Strauss process
`ppm(X ~ x, Strauss(0.1))`	Strauss process with conditional intensity loglinear in x

It is also possible to fit models that depend on other covariates.

Manipulating the fitted model:

`plot.ppm`	Plot the fitted model
`predict.ppm`	Compute the spatial trend and conditional intensity of the fitted point process model
`coef.ppm`	Extract the fitted model coefficients
`parameters`	Extract all model parameters
`formula.ppm`	Extract the trend formula
`intensity.ppm`	Compute fitted intensity
`Kmodel.ppm`	K function of fitted model
`pcfmodel.ppm`	pair correlation of fitted model
`fitted.ppm`	Compute fitted conditional intensity at quadrature points
`residuals.ppm`	Compute point process residuals at quadrature points
`update.ppm`	Update the fit
`vcov.ppm`	Variance-covariance matrix of estimates
`rmh.ppm`	Simulate from fitted model
`simulate.ppm`	Simulate from fitted model
`print.ppm`	Print basic information about a fitted model
`summary.ppm`	Summarise a fitted model
`effectfun`	Compute the fitted effect of one covariate
`logLik.ppm`	log-likelihood or log-pseudolikelihood
`anova.ppm`	Analysis of deviance
`model.frame.ppm`	Extract data frame used to fit model
`model.images`	Extract spatial data used to fit model
`model.depends`	Identify variables in the model
`as.interact`	Interpoint interaction component of model
`fitin`	Extract fitted interpoint interaction
`is.hybrid`	Determine whether the model is a hybrid
`valid.ppm`	Check the model is a valid point process
`project.ppm`	Ensure the model is a valid point process

For model selection, you can also use the generic functions `step`, `drop1` and `AIC` on fitted point process models.

See `spatstat.options` to control plotting of fitted model.

To specify a point process model:

The first order "trend" of the model is determined by an R language formula. The formula specifies the form of the *logarithm* of the trend.

`X ~ 1`	No trend (stationary)
`X ~ x`	Loglinear trend $\lambda(x, y) = \exp(\alpha + \beta x)$ where x, y are Cartesian coordinates
`X ~ polynom(x,y,3)`	Log-cubic polynomial trend
`X ~ harmonic(x,y,2)`	Log-harmonic polynomial trend
`X ~ Z`	Loglinear function of covariate `Z` $\lambda(x, y) = \exp(\alpha + \beta Z(x, y))$

The higher order ("interaction") components are described by an object of class `"interact"`. Such objects are created by:

`Poisson()`	the Poisson point process
`AreaInter()`	Area-interaction process
`BadGey()`	multiscale Geyer process
`Concom()`	connected component interaction
`DiggleGratton()`	Diggle-Gratton potential
`DiggleGatesStibbard()`	Diggle-Gates-Stibbard potential
`Fiksel()`	Fiksel pairwise interaction process
`Geyer()`	Geyer's saturation process
`Hardcore()`	Hard core process
`HierHard()`	Hierarchical multiype hard core process
`HierStrauss()`	Hierarchical multiype Strauss process
`HierStraussHard()`	Hierarchical multiype Strauss-hard core process
`Hybrid()`	Hybrid of several interactions
`LennardJones()`	Lennard-Jones potential
`MultiHard()`	multitype hard core process
`MultiStrauss()`	multitype Strauss process
`MultiStraussHard()`	multitype Strauss/hard core process
`OrdThresh()`	Ord process, threshold potential
`Ord()`	Ord model, user-supplied potential
`PairPiece()`	pairwise interaction, piecewise constant
`Pairwise()`	pairwise interaction, user-supplied potential
`Penttinen()`	Penttinen pairwise interaction
`SatPiece()`	Saturated pair model, piecewise constant potential
`Saturated()`	Saturated pair model, user-supplied potential
`Softcore()`	pairwise interaction, soft core potential
`Strauss()`	Strauss process
`StraussHard()`	Strauss/hard core point process
`Triplets()`	Geyer triplets process

Note that it is also possible to combine several such interactions using `Hybrid`.

Finer control over model fitting:

A quadrature scheme is represented by an object of class `"quad"`. To create a quadrature scheme, typically use `quadscheme`.

quadscheme	default quadrature scheme using rectangular cells or Dirichlet cells
pixelquad	quadrature scheme based on image pixels
quad	create an object of class "quad"

To inspect a quadrature scheme:

plot(Q)	plot quadrature scheme Q
print(Q)	print basic information about quadrature scheme Q
summary(Q)	summary of quadrature scheme Q

A quadrature scheme consists of data points, dummy points, and weights. To generate dummy points:

default.dummy	default pattern of dummy points
gridcentres	dummy points in a rectangular grid
rstrat	stratified random dummy pattern
spokes	radial pattern of dummy points
corners	dummy points at corners of the window

To compute weights:

gridweights	quadrature weights by the grid-counting rule
dirichletWeights	quadrature weights are Dirichlet tile areas

Simulation and goodness-of-fit for fitted models:

rmh.ppm	simulate realisations of a fitted model
simulate.ppm	simulate realisations of a fitted model
envelope	compute simulation envelopes for a fitted model

Point process models on a linear network:

An object of class "lpp" represents a pattern of points on a linear network. Point process models can also be fitted to these objects. Currently only Poisson models can be fitted.

lppm	point process model on linear network
anova.lppm	analysis of deviance for point process model on linear network
envelope.lppm	simulation envelopes for point process model on linear network
fitted.lppm	fitted intensity values
predict.lppm	model prediction on linear network
linim	pixel image on linear network
plot.linim	plot a pixel image on linear network
eval.linim	evaluate expression involving images
linfun	function defined on linear network
methods.linfun	conversion facilities

V. MODEL FITTING (DETERMINANTAL POINT PROCESS MODELS)

Code for fitting *determinantal point process models* has recently been added to **spatstat**.

For information, see the help file for dppm.

VI. MODEL FITTING (SPATIAL LOGISTIC REGRESSION)

Logistic regression

Pixel-based spatial logistic regression is an alternative technique for analysing spatial point patterns that is widely used in Geographical Information Systems. It is approximately equivalent to fitting a Poisson point process model.

In pixel-based logistic regression, the spatial domain is divided into small pixels, the presence or absence of a data point in each pixel is recorded, and logistic regression is used to model the presence/absence indicators as a function of any covariates.

Facilities for performing spatial logistic regression are provided in **spatstat** for comparison purposes.

Fitting a spatial logistic regression

Spatial logistic regression is performed by the function `slrm`. Its result is an object of class `"slrm"`. There are many methods for this class, including methods for `print`, `fitted`, `predict`, `simulate`, `anova`, `coef`, `logLik`, `terms`, `update`, `formula` and `vcov`.

For example, if `X` is a point pattern (class `"ppp"`):

command	*model*
`slrm(X ~ 1)`	Complete Spatial Randomness
`slrm(X ~ x)`	Poisson process with intensity loglinear in x coordinate
`slrm(X ~ Z)`	Poisson process with intensity loglinear in covariate `Z`

Manipulating a fitted spatial logistic regression

`anova.slrm`	Analysis of deviance
`coef.slrm`	Extract fitted coefficients
`vcov.slrm`	Variance-covariance matrix of fitted coefficients
`fitted.slrm`	Compute fitted probabilities or intensity
`logLik.slrm`	Evaluate loglikelihood of fitted model
`plot.slrm`	Plot fitted probabilities or intensity
`predict.slrm`	Compute predicted probabilities or intensity with new data
`simulate.slrm`	Simulate model

There are many other undocumented methods for this class, including methods for `print`, `update`, `formula` and `terms`. Stepwise model selection is possible using `step` or `stepAIC`.

VII. SIMULATION

There are many ways to generate a random point pattern, line segment pattern, pixel image or tessellation in **spatstat**.

Random point patterns:

`runifpoint`	generate n independent uniform random points
`rpoint`	generate n independent random points
`rmpoint`	generate n independent multitype random points
`rpoispp`	simulate the (in)homogeneous Poisson point process

rmpoispp	simulate the (in)homogeneous multitype Poisson point process
runifdisc	generate n independent uniform random points in disc
rstrat	stratified random sample of points
rsyst	systematic random sample (grid) of points
rMaternI	simulate the Matérn Model I inhibition process
rMaternII	simulate the Matérn Model II inhibition process
rSSI	simulate Simple Sequential Inhibition process
rHardcore	simulate hard core process (perfect simulation)
rStrauss	simulate Strauss process (perfect simulation)
rStraussHard	simulate Strauss-hard core process (perfect simulation)
rDiggleGratton	simulate Diggle-Gratton process (perfect simulation)
rDGS	simulate Diggle-Gates-Stibbard process (perfect simulation)
rPenttinen	simulate Penttinen process (perfect simulation)
rNeymanScott	simulate a general Neyman-Scott process
rMatClust	simulate the Matérn Cluster process
rThomas	simulate the Thomas process
rLGCP	simulate the log-Gaussian Cox process
rGaussPoisson	simulate the Gauss-Poisson cluster process
rCauchy	simulate Neyman-Scott process with Cauchy clusters
rVarGamma	simulate Neyman-Scott process with Variance Gamma clusters
rcell	simulate the Baddeley-Silverman cell process
runifpointOnLines	generate n random points along specified line segments
rpoisppOnLines	generate Poisson random points along specified line segments

Resampling a point pattern:

quadratresample	block resampling
rjitter	apply random displacements to points in a pattern
rshift	random shifting of (subsets of) points
rthin	random thinning

See also varblock for estimating the variance of a summary statistic by block resampling, and lohboot for another bootstrap technique.

Fitted point process models:

If you have fitted a point process model to a point pattern dataset, the fitted model can be simulated.

Cluster process models are fitted by the function kppm yielding an object of class "kppm". To generate one or more simulated realisations of this fitted model, use simulate.kppm.

Gibbs point process models are fitted by the function ppm yielding an object of class "ppm". To generate a simulated realisation of this fitted model, use rmh. To generate one or more simulated realisations of the fitted model, use simulate.ppm.

Other random patterns:

rlinegrid	generate a random array of parallel lines through a window
rpoisline	simulate the Poisson line process within a window
rpoislinetess	generate random tessellation using Poisson line process
rMosaicSet	generate random set by selecting some tiles of a tessellation
rMosaicField	generate random pixel image by assigning random values in each tile of a tessellation

Simulation-based inference

envelope	critical envelope for Monte Carlo test of goodness-of-fit

qqplot.ppm	diagnostic plot for interpoint interaction
scan.test	spatial scan statistic/test
studpermu.test	studentised permutation test
segregation.test	test of segregation of types

VIII. TESTS AND DIAGNOSTICS

Hypothesis tests:

quadrat.test	χ^2 goodness-of-fit test on quadrat counts
clarkevans.test	Clark and Evans test
cdf.test	Spatial distribution goodness-of-fit test
berman.test	Berman's goodness-of-fit tests
envelope	critical envelope for Monte Carlo test of goodness-of-fit
scan.test	spatial scan statistic/test
dclf.test	Diggle-Cressie-Loosmore-Ford test
mad.test	Mean Absolute Deviation test
anova.ppm	Analysis of Deviance for point process models

More recently-developed tests:

dg.test	Dao-Genton test
bits.test	Balanced independent two-stage test
dclf.progress	Progress plot for DCLF test
mad.progress	Progress plot for MAD test

Sensitivity diagnostics:

Classical measures of model sensitivity such as leverage and influence have been adapted to point process models.

leverage.ppm	Leverage for point process model
influence.ppm	Influence for point process model
dfbetas.ppm	Parameter influence

Diagnostics for covariate effect:

Classical diagnostics for covariate effects have been adapted to point process models.

parres	Partial residual plot
addvar	Added variable plot
rhohat	Kernel estimate of covariate effect
rho2hat	Kernel estimate of covariate effect (bivariate)

Residual diagnostics:

Residuals for a fitted point process model, and diagnostic plots based on the residuals, were introduced in Baddeley et al (2005) and Baddeley, Rubak and Møller (2011).

Type `demo(diagnose)` for a demonstration of the diagnostics features.

diagnose.ppm	diagnostic plots for spatial trend
qqplot.ppm	diagnostic Q-Q plot for interpoint interaction

residualspaper	examples from Baddeley et al (2005)
Kcom	model compensator of K function
Gcom	model compensator of G function
Kres	score residual of K function
Gres	score residual of G function
psst	pseudoscore residual of summary function
psstA	pseudoscore residual of empty space function
psstG	pseudoscore residual of G function
compareFit	compare compensators of several fitted models

Resampling and randomisation procedures

You can build your own tests based on randomisation and resampling using the following capabilities:

quadratresample	block resampling
rjitter	apply random displacements to points in a pattern
rshift	random shifting of (subsets of) points
rthin	random thinning

IX. DOCUMENTATION

The online manual entries are quite detailed and should be consulted first for information about a particular function.

The book Baddeley, Rubak and Turner (2015) is a complete course on analysing spatial point patterns, with full details about **spatstat**.

Older material (which is now out-of-date but is freely available) includes Baddeley and Turner (2005a), a brief overview of the package in its early development; Baddeley and Turner (2005b), a more detailed explanation of how to fit point process models to data; and Baddeley (2010), a complete set of notes from a 2-day workshop on the use of **spatstat**.

Type `citation("spatstat")` to get a list of these references.

Licence

Acknowledgements

Kasper Klitgaard Berthelsen, Ottmar Cronie, Yongtao Guan, Ute Hahn, Abdollah Jalilian, Marie-Colette van Lieshout, Greg McSwiggan, Tuomas Rajala, Suman Rakshit, Dominic Schuhmacher, Rasmus Waagepetersen and Hangsheng Wang made substantial contributions of code.

Additional contributions and suggestions from Monsuru Adepeju, Corey Anderson, Ang Qi Wei, Marcel Austenfeld, Sandro Azaele, Malissa Baddeley, Guy Bayegnak, Colin Beale, Melanie Bell, Thomas Bendtsen, Ricardo Bernhardt, Andrew Bevan, Brad Biggerstaff, Anders Bilgrau, Leanne Bischof, Christophe Biscio, Roger Bivand, Jose M. Blanco Moreno, Florent Bonneu, Julian Burgos, Simon Byers, Ya-Mei Chang, Jianbao Chen, Igor Chernayavsky, Y.C. Chin, Bjarke Christensen, Jean-Francois Coeurjolly, Kim Colyvas, Rochelle Constantine, Robin Corria Ainslie, Richard Cotton, Marcelino de la Cruz, Peter Dalgaard, Mario D'Antuono, Sourav Das, Tilman Davies, Peter Diggle, Patrick Donnelly, Ian Dryden, Stephen Eglen, Ahmed El-Gabbas, Belarmain Fandohan, Olivier Flores, David Ford, Peter Forbes, Shane Frank, Janet Franklin, Funwi-Gabga Neba, Oscar Garcia, Agnes Gault, Jonas Geldmann, Marc Genton, Shaaban Ghalandarayeshi, Julian Gilbey,

Jason Goldstick, Pavel Grabarnik, C. Graf, Ute Hahn, Andrew Hardegen, Martin Bøgsted Hansen, Martin Hazelton, Juha Heikkinen, Mandy Hering, Markus Herrmann, Paul Hewson, Kassel Hingee, Kurt Hornik, Philipp Hunziker, Jack Hywood, Ross Ihaka, Čenk Içös, Aruna Jammalamadaka, Robert John-Chandran, Devin Johnson, Mahdieh Khanmohammadi, Bob Klaver, Lily Kozmian-Ledward, Peter Kovesi, Mike Kuhn, Jeff Laake, Frederic Lavancier, Tom Lawrence, Robert Lamb, Jonathan Lee, George Leser, Li Haitao, George Limitsios, Andrew Lister, Ben Madin, Martin Maechler, Kiran Marchikanti, Jeff Marcus, Robert Mark, Peter McCullagh, Monia Mahling, Jorge Mateu Mahiques, Ulf Mehlig, Frederico Mestre, Sebastian Wastl Meyer, Mi Xiangcheng, Lore De Middeleer, Robin Milne, Enrique Miranda, Jesper Møller, Mehdi Moradi, Virginia Morera Pujol, Erika Mudrak, Gopalan Nair, Nader Najari, Nicoletta Nava, Linda Stougaard Nielsen, Felipe Nunes, Jens Randel Nyengaard, Jens Oehlschlägel, Thierry Onkelinx, Sean O'Riordan, Evgeni Parilov, Jeff Picka, Nicolas Picard, Mike Porter, Sergiy Protsiv, Adrian Raftery, Suman Rakshit, Ben Ramage, Pablo Ramon, Xavier Raynaud, Nicholas Read, Matt Reiter, Ian Renner, Tom Richardson, Brian Ripley, Ted Rosenbaum, Barry Rowlingson, Jason Rudokas, John Rudge, Christopher Ryan, Farzaneh Safavimanesh, Aila Särkkä, Cody Schank, Katja Schladitz, Sebastian Schutte, Bryan Scott, Olivia Semboli, François Sémécurbe, Vadim Shcherbakov, Shen Guochun, Shi Peijian, Harold-Jeffrey Ship, Tammy L Silva, Ida-Maria Sintorn, Yong Song, Malte Spiess, Mark Stevenson, Kaspar Stucki, Michael Sumner, P. Surovy, Ben Taylor, Thordis Linda Thorarinsdottir, Leigh Torres, Berwin Turlach, Torben Tvedebrink, Kevin Ummer, Medha Uppala, Andrew van Burgel, Tobias Verbeke, Mikko Vihtakari, Alexendre Villers, Fabrice Vinatier, Sasha Voss, Sven Wagner, Hao Wang, H. Wendrock, Jan Wild, Carl G. Witthoft, Selene Wong, Maxime Woringer, Mike Zamboni and Achim Zeileis.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A. (2010) *Analysing spatial point patterns in R*. Workshop notes, Version 4.1. Online technical publication, CSIRO. https://research.csiro.au/software/wp-content/uploads/sites/6/2015/02/Rspatialcourse_CMIS_PDF-Standard.pdf

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

Baddeley, A. and Turner, R. (2005a) Spatstat: an R package for analyzing spatial point patterns. *Journal of Statistical Software* **12**:6, 1–42. URL: www.jstatsoft.org, ISSN: 1548-7660.

Baddeley, A. and Turner, R. (2005b) Modelling spatial point patterns in R. In: A. Baddeley, P. Gregori, J. Mateu, R. Stoica, and D. Stoyan, editors, *Case Studies in Spatial Point Pattern Modelling*, Lecture Notes in Statistics number 185. Pages 23–74. Springer-Verlag, New York, 2006. ISBN: 0-387-28311-0.

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

Baddeley, A., Turner, R., Mateu, J. and Bevan, A. (2013) Hybrids of Gibbs point process models and their implementation. *Journal of Statistical Software* **55**:11, 1–43. http://www.jstatsoft.org/v55/i11/

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

Diggle, P.J. (2014) *Statistical Analysis of Spatial and Spatio-Temporal Point Patterns*, Third edition. Chapman and Hall/CRC.

Gelfand, A.E., Diggle, P.J., Fuentes, M. and Guttorp, P., editors (2010) *Handbook of Spatial Statistics*. CRC Press.

Huang, F. and Ogata, Y. (1999) Improvements of the maximum pseudo-likelihood estimators in various spatial statistical models. *Journal of Computational and Graphical Statistics* **8**, 510–530.

Illian, J., Penttinen, A., Stoyan, H. and Stoyan, D. (2008) *Statistical Analysis and Modelling of Spatial Point Patterns.* Wiley.

Waagepetersen, R. An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63** (2007) 252–258.

`adaptive.density` *Intensity Estimate of Point Pattern Using Tessellation*

Description

Computes an adaptive estimate of the intensity function of a point pattern.

Usage

```
adaptive.density(X, f = 0.1, ..., nrep = 1, verbose=TRUE)
```

Arguments

`X`	Point pattern dataset (object of class "ppp").
`f`	Fraction (between 0 and 1 inclusive) of the data points that will be removed from the data and used to determine a tessellation for the intensity estimate.
`...`	Arguments passed to `as.im` determining the pixel resolution of the result.
`nrep`	Number of independent repetitions of the randomised procedure.
`verbose`	Logical value indicating whether to print progress reports.

Details

This function is an alternative to `density.ppp`. It computes an estimate of the intensity function of a point pattern dataset. The result is a pixel image giving the estimated intensity,

If `f=1`, the Voronoi estimate (Barr and Schoenberg, 2010) is computed: the point pattern `X` is used to construct a Voronoi/Dirichlet tessellation (see `dirichlet`); the areas of the Dirichlet tiles are computed; the estimated intensity in each tile is the reciprocal of the tile area.

If `f=0`, the intensity estimate at every location is equal to the average intensity (number of points divided by window area).

If `f` is strictly between 0 and 1, the dataset `X` is randomly split into two patterns `A` and `B` containing a fraction `f` and `1-f`, respectively, of the original data. The subpattern `A` is used to construct a Dirichlet tessellation, while the subpattern `B` is retained for counting. For each tile of the Dirichlet tessellation, we count the number of points of `B` falling in the tile, and divide by the area of the same tile, to obtain an estimate of the intensity of the pattern `B` in the tile. This estimate is divided by `1-f` to obtain an estimate of the intensity of `X` in the tile. The result is a pixel image of intensity estimates which are constant on each tile of the tessellation.

If `nrep` is greater than 1, this randomised procedure is repeated `nrep` times, and the results are averaged.

This technique has been used by Ogata et al. (2003), Ogata (2004) and Baddeley (2007).

Value

A pixel image (object of class "im") whose values are estimates of the intensity of X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. (2007) Validation of statistical models for spatial point patterns. In J.G. Babu and E.D. Feigelson (eds.) *SCMA IV: Statistical Challenges in Modern Astronomy IV*, volume 317 of Astronomical Society of the Pacific Conference Series, San Francisco, California USA, 2007. Pages 22–38.

Barr, C., and Schoenberg, F.P. (2010). On the Voronoi estimator for the intensity of an inhomogeneous planar Poisson process. *Biometrika* **97** (4), 977–984.

Ogata, Y. (2004) Space-time model for regional seismicity and detection of crustal stress changes. *Journal of Geophysical Research*, **109**, 2004.

Ogata, Y., Katsura, K. and Tanemura, M. (2003). Modelling heterogeneous space-time occurrences of earthquakes and its residual analysis. *Applied Statistics* **52** 499–509.

See Also

density.ppp, dirichlet, im.object.

Examples

```
plot(adaptive.density(nztrees, 1), main="Voronoi estimate")
nr <- if(interactive()) 100 else 5
plot(adaptive.density(nztrees, nrep=nr), main="Adaptive estimate")
```

---

add.texture *Fill Plot With Texture*

Description

Draws a simple texture inside a region on the plot.

Usage

```
add.texture(W, texture = 4, spacing = NULL, ...)
```

Arguments

W	Window (object of class "owin") inside which the texture should be drawn.
texture	Integer from 1 to 8 identifying the type of texture. See Details.
spacing	Spacing between elements of the texture, in units of the current plot.
...	Further arguments controlling the plot colour, line width etc.

Details

The chosen texture, confined to the window `W`, will be added to the current plot. The available textures are:

texture=1: Small crosses arranged in a square grid.

texture=2: Parallel vertical lines.

texture=3: Parallel horizontal lines.

texture=4: Parallel diagonal lines at 45 degrees from the horizontal.

texture=5: Parallel diagonal lines at 135 degrees from the horizontal.

texture=6: Grid of horizontal and vertical lines.

texture=7: Grid of diagonal lines at 45 and 135 degrees from the horizontal.

texture=8: Grid of hexagons.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`owin`, `plot.owin`, `textureplot`, `texturemap`.

Examples

```
W <- Window(chorley)
plot(W, main="")
add.texture(W, 7)
```

`addvar` *Added Variable Plot for Point Process Model*

Description

Computes the coordinates for an Added Variable Plot for a fitted point process model.

Usage

```
addvar(model, covariate, ...,
                     subregion=NULL,
                     bw="nrd0", adjust=1,
                     from=NULL, to=NULL, n=512,
                     bw.input = c("points", "quad"),
                     bw.restrict = FALSE,
                     covname, crosscheck=FALSE)
```

Arguments

`model`	Fitted point process model (object of class "ppm").
`covariate`	The covariate to be added to the model. Either a pixel image, a `function(x,y)`, or a character string giving the name of a covariate that was supplied when the model was fitted.
`subregion`	Optional. A window (object of class "owin") specifying a subset of the spatial domain of the data. The calculation will be confined to the data in this subregion.
`bw`	Smoothing bandwidth or bandwidth rule (passed to `density.default`).
`adjust`	Smoothing bandwidth adjustment factor (passed to `density.default`).
`n, from, to`	Arguments passed to `density.default` to control the number and range of values at which the function will be estimated.
`...`	Additional arguments passed to `density.default`.
`bw.input`	Character string specifying the input data used for automatic bandwidth selection.
`bw.restrict`	Logical value, specifying whether bandwidth selection is performed using data from the entire spatial domain or from the `subregion`.
`covname`	Optional. Character string to use as the name of the covariate.
`crosscheck`	For developers only. Logical value indicating whether to perform cross-checks on the validity of the calculation.

Details

This command generates the plot coordinates for an Added Variable Plot for a spatial point process model.

Added Variable Plots (Cox, 1958, sec 4.5; Wang, 1985) are commonly used in linear models and generalized linear models, to decide whether a model with response y and predictors x would be improved by including another predictor z.

In a (generalised) linear model with response y and predictors x, the Added Variable Plot for a new covariate z is a plot of the smoothed Pearson residuals from the original model against the scaled residuals from a weighted linear regression of z on x. If this plot has nonzero slope, then the new covariate z is needed. For general advice see Cook and Weisberg(1999); Harrell (2001).

Essentially the same technique can be used for a spatial point process model (Baddeley et al, 2012).

The argument `model` should be a fitted spatial point process model (object of class "ppm").

The argument `covariate` identifies the covariate that is to be considered for addition to the model. It should be either a pixel image (object of class "im") or a `function(x,y)` giving the values of the covariate at any spatial location. Alternatively `covariate` may be a character string, giving the name of a covariate that was supplied (in the `covariates` argument to `ppm`) when the model was fitted, but was not used in the model.

The result of `addvar(model, covariate)` is an object belonging to the classes "addvar" and "fv". Plot this object to generate the added variable plot.

Note that the plot method shows the pointwise significance bands for a test of the *null* model, i.e. the null hypothesis that the new covariate has no effect.

The smoothing bandwidth is controlled by the arguments `bw`, `adjust`, `bw.input` and `bw.restrict`. If `bw` is a numeric value, then the bandwidth is taken to be `adjust * bw`. If `bw` is a string representing a bandwidth selection rule (recognised by `density.default`) then the bandwidth is selected by this rule.

The data used for automatic bandwidth selection are specified by bw.input and bw.restrict. If bw.input="points" (the default) then bandwidth selection is based on the covariate values at the points of the original point pattern dataset to which the model was fitted. If bw.input="quad" then bandwidth selection is based on the covariate values at every quadrature point used to fit the model. If bw.restrict=TRUE then the bandwidth selection is performed using only data from inside the subregion.

Value

An object of class "addvar" containing the coordinates for the added variable plot. There is a plot method.

Slow computation

In a large dataset, computation can be very slow if the default settings are used, because the smoothing bandwidth is selected automatically. To avoid this, specify a numerical value for the bandwidth bw. One strategy is to use a coarser subset of the data to select bw automatically. The selected bandwidth can be read off the print output for addvar.

Internal data

The return value has an attribute "spatial" which contains the internal data: the computed values of the residuals, and of all relevant covariates, at each quadrature point of the model. It is an object of class "ppp" with a data frame of marks.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz>, Ya-Mei Chang and Yong Song.

References

Baddeley, A., Chang, Y.-M., Song, Y. and Turner, R. (2013) Residual diagnostics for covariate effects in spatial point process models. *Journal of Computational and Graphical Statistics*, **22**, 886–905.

Cook, R.D. and Weisberg, S. (1999) *Applied regression, including computing and graphics*. New York: Wiley.

Cox, D.R. (1958) *Planning of Experiments*. New York: Wiley.

Harrell, F. (2001) *Regression Modeling Strategies*. New York: Springer.

Wang, P. (1985) Adding a variable in generalized linear models. *Technometrics* **27**, 273–276.

See Also

parres, rhohat, rho2hat.

Examples

```
X <-  rpoispp(function(x,y){exp(3+3*x)})
model <- ppm(X, ~y)
adv <- addvar(model, "x")
plot(adv)
adv <- addvar(model, "x", subregion=square(0.5))
```

`affine` *Apply Affine Transformation*

Description

Applies any affine transformation of the plane (linear transformation plus vector shift) to a plane geometrical object, such as a point pattern or a window.

Usage

```
affine(X, ...)
```

Arguments

`X`	Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class `"ppp"`), a line segment pattern (object of class `"psp"`), a window (object of class `"owin"`) or a pixel image (object of class `"im"`).
`...`	Arguments determining the affine transformation.

Details

This is generic. Methods are provided for point patterns (`affine.ppp`) and windows (`affine.owin`).

Value

Another object of the same type, representing the result of applying the affine transformation.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`affine.ppp`, `affine.psp`, `affine.owin`, `affine.im`, `flipxy`, `reflect`, `rotate`, `shift`

`affine.im` *Apply Affine Transformation To Pixel Image*

Description

Applies any affine transformation of the plane (linear transformation plus vector shift) to a pixel image.

Usage

```
  ## S3 method for class 'im'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)
```

Arguments

`X`	Pixel image (object of class `"im"`).
`mat`	Matrix representing a linear transformation.
`vec`	Vector of length 2 representing a translation.
`...`	Optional arguments passed to `as.mask` controlling the pixel resolution of the transformed image.

Details

The image is subjected first to the linear transformation represented by `mat` (multiplying on the left by `mat`), and then the result is translated by the vector `vec`.

The argument `mat` must be a nonsingular 2×2 matrix.

This is a method for the generic function `affine`.

Value

Another pixel image (of class `"im"`) representing the result of applying the affine transformation.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`affine`, `affine.ppp`, `affine.psp`, `affine.owin`, `rotate`, `shift`

Examples

```
X <- setcov(owin())
stretch <- diag(c(2,3))
Y <- affine(X, mat=stretch)
shear <- matrix(c(1,0,0.6,1),ncol=2, nrow=2)
Z <- affine(X, mat=shear)
```

---

affine.linnet *Apply Geometrical Transformations to a Linear Network*

Description

Apply geometrical transformations to a linear network.

Usage

```
## S3 method for class 'linnet'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)

## S3 method for class 'linnet'
shift(X, vec=c(0,0), ..., origin=NULL)

## S3 method for class 'linnet'
rotate(X, angle=pi/2, ..., centre=NULL)

## S3 method for class 'linnet'
scalardilate(X, f, ...)

## S3 method for class 'linnet'
rescale(X, s, unitname)
```

Arguments

X	Linear network (object of class "linnet").
mat	Matrix representing a linear transformation.
vec	Vector of length 2 representing a translation.
angle	Rotation angle in radians.
f	Scalar dilation factor.
s	Unit conversion factor: the new units are s times the old units.
...	Arguments passed to other methods.
origin	Character string determining a location that will be shifted to the origin. Options are "centroid", "midpoint" and "bottomleft". Partially matched.
centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).
unitname	Optional. New name for the unit of length. A value acceptable to the function unitname<-

Details

These functions are methods for the generic functions affine, shift, rotate, rescale and scalardilate applicable to objects of class "linnet".

All of these functions perform geometrical transformations on the object X, except for rescale, which simply rescales the units of length.

Value

Another linear network (of class "linnet") representing the result of applying the geometrical transformation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

linnet and as.linnet.

Generic functions affine, shift, rotate, scalardilate, rescale.

Examples

```
U <- rotate(simplenet, pi)
stretch <- diag(c(2,3))
Y <- affine(simplenet, mat=stretch)
shear <- matrix(c(1,0,0.6,1),ncol=2, nrow=2)
Z <- affine(simplenet, mat=shear, vec=c(0, 1))
```

---

affine.lpp *Apply Geometrical Transformations to Point Pattern on a Linear Network*

Description

Apply geometrical transformations to a point pattern on a linear network.

Usage

```
## S3 method for class 'lpp'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)

## S3 method for class 'lpp'
shift(X, vec=c(0,0), ..., origin=NULL)

## S3 method for class 'lpp'
rotate(X, angle=pi/2, ..., centre=NULL)

## S3 method for class 'lpp'
scalardilate(X, f, ...)

## S3 method for class 'lpp'
rescale(X, s, unitname)
```

Arguments

X	Point pattern on a linear network (object of class "lpp").
mat	Matrix representing a linear transformation.
vec	Vector of length 2 representing a translation.
angle	Rotation angle in radians.
f	Scalar dilation factor.
s	Unit conversion factor: the new units are s times the old units.
...	Arguments passed to other methods.
origin	Character string determining a location that will be shifted to the origin. Options are "centroid", "midpoint" and "bottomleft". Partially matched.

centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).
unitname	Optional. New name for the unit of length. A value acceptable to the function unitname<-

Details

These functions are methods for the generic functions affine, shift, rotate, rescale and scalardilate applicable to objects of class "lpp".

All of these functions perform geometrical transformations on the object X, except for rescale, which simply rescales the units of length.

Value

Another point pattern on a linear network (object of class "lpp") representing the result of applying the geometrical transformation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

lpp.

Generic functions affine, shift, rotate, scalardilate, rescale.

Examples

```
X <- rpoislpp(2, simplenet)
U <- rotate(X, pi)
stretch <- diag(c(2,3))
Y <- affine(X, mat=stretch)
shear <- matrix(c(1,0,0.6,1),ncol=2, nrow=2)
Z <- affine(X, mat=shear, vec=c(0, 1))
```

---

affine.owin *Apply Affine Transformation To Window*

Description

Applies any affine transformation of the plane (linear transformation plus vector shift) to a window.

Usage

```
  ## S3 method for class 'owin'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ..., rescue=TRUE)
```

Arguments

X	Window (object of class "owin").
mat	Matrix representing a linear transformation.
vec	Vector of length 2 representing a translation.
rescue	Logical. If TRUE, the transformed window will be processed by rescue.rectangle.
...	Optional arguments passed to as.mask controlling the pixel resolution of the transformed window, if X is a binary pixel mask.

Details

The window is subjected first to the linear transformation represented by mat (multiplying on the left by mat), and then the result is translated by the vector vec.

The argument mat must be a nonsingular 2×2 matrix.

This is a method for the generic function affine.

Value

Another window (of class "owin") representing the result of applying the affine transformation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

affine, affine.ppp, affine.psp, affine.im, rotate, shift

Examples

```
  # shear transformation
  shear <- matrix(c(1,0,0.6,1),ncol=2)
  X <- affine(owin(), shear)
  ## Not run:
  plot(X)

## End(Not run)
  data(letterR)
  affine(letterR, shear, c(0, 0.5))
  affine(as.mask(letterR), shear, c(0, 0.5))
```

---

affine.ppp   *Apply Affine Transformation To Point Pattern*

Description

Applies any affine transformation of the plane (linear transformation plus vector shift) to a point pattern.

Usage

```
 ## S3 method for class 'ppp'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)
```

Arguments

`X`	Point pattern (object of class "ppp").
`mat`	Matrix representing a linear transformation.
`vec`	Vector of length 2 representing a translation.
`...`	Arguments passed to `affine.owin` affecting the handling of the observation window, if it is a binary pixel mask.

Details

The point pattern, and its window, are subjected first to the linear transformation represented by `mat` (multiplying on the left by `mat`), and are then translated by the vector `vec`.

The argument `mat` must be a nonsingular 2×2 matrix.

This is a method for the generic function `affine`.

Value

Another point pattern (of class "ppp") representing the result of applying the affine transformation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`affine`, `affine.owin`, `affine.psp`, `affine.im`, `flipxy`, `rotate`, `shift`

Examples

```
  data(cells)
  # shear transformation
  X <- affine(cells, matrix(c(1,0,0.6,1),ncol=2))
  ## Not run:
  plot(X)
  # rescale y coordinates by factor 1.3
  plot(affine(cells, diag(c(1,1.3))))

## End(Not run)
```

affine.psp *Apply Affine Transformation To Line Segment Pattern*

Description

Applies any affine transformation of the plane (linear transformation plus vector shift) to a line segment pattern.

Usage

```
## S3 method for class 'psp'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)
```

Arguments

X	Line Segment pattern (object of class "psp").
mat	Matrix representing a linear transformation.
vec	Vector of length 2 representing a translation.
...	Arguments passed to affine.owin affecting the handling of the observation window, if it is a binary pixel mask.

Details

The line segment pattern, and its window, are subjected first to the linear transformation represented by mat (multiplying on the left by mat), and are then translated by the vector vec.

The argument mat must be a nonsingular 2×2 matrix.

This is a method for the generic function affine.

Value

Another line segment pattern (of class "psp") representing the result of applying the affine transformation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

affine, affine.owin, affine.ppp, affine.im, flipxy, rotate, shift

Examples

```
oldpar <- par(mfrow=c(2,1))
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(X, main="original")
# shear transformation
Y <- affine(X, matrix(c(1,0,0.6,1),ncol=2))
plot(Y, main="transformed")
par(oldpar)
```

```
#
# rescale y coordinates by factor 0.2
affine(X, diag(c(1,0.2)))
```

affine.tess *Apply Geometrical Transformation To Tessellation*

Description

Apply various geometrical transformations of the plane to each tile in a tessellation.

Usage

```
  ## S3 method for class 'tess'
reflect(X)

  ## S3 method for class 'tess'
shift(X, ...)

  ## S3 method for class 'tess'
rotate(X, angle=pi/2, ..., centre=NULL)

  ## S3 method for class 'tess'
scalardilate(X, f, ...)

  ## S3 method for class 'tess'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)
```

Arguments

X	Tessellation (object of class "tess").
angle	Rotation angle in radians (positive values represent anticlockwise rotations).
mat	Matrix representing a linear transformation.
vec	Vector of length 2 representing a translation.
f	Positive number giving scale factor.
...	Arguments passed to other methods.
centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).

Details

These are method for the generic functions reflect, shift, rotate, scalardilate, affine for tessellations (objects of class "tess").

The individual tiles of the tessellation, and the window containing the tessellation, are all subjected to the same geometrical transformation.

The transformations are performed by the corresponding method for windows (class "owin") or images (class "im") depending on the type of tessellation.

If the argument origin is used in shift.tess it is interpreted as applying to the window containing the tessellation. Then all tiles are shifted by the same vector.

Value

Another tessellation (of class "tess") representing the result of applying the geometrical transformation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Generic functions reflect, shift, rotate, scalardilate, affine.

Methods for windows: reflect.default, shift.owin, rotate.owin, scalardilate.owin, affine.owin.

Methods for images: reflect.im, shift.im, rotate.im, scalardilate.im, affine.im.

Examples

```
live <- interactive()
if(live) {
  H <- hextess(letterR, 0.2)
  plot(H)
  plot(reflect(H))
  plot(rotate(H, pi/3))
} else H <- hextess(letterR, 0.6)

# shear transformation
shear <- matrix(c(1,0,0.6,1),2,2)
sH <- affine(H, shear)
if(live) plot(sH)
```

---

allstats *Calculate four standard summary functions of a point pattern.*

Description

Calculates the F, G, J, and K summary functions for an unmarked point pattern. Returns them as a function array (of class "fasp", see fasp.object).

Usage

```
allstats(pp, ..., dataname=NULL, verb=FALSE)
```

Arguments

pp	The observed point pattern, for which summary function estimates are required. An object of class "ppp". It must not be marked.
...	Optional arguments passed to the summary functions Fest, Gest, Jest and Kest.
dataname	A character string giving an optional (alternative) name for the point pattern.
verb	A logical value meaning “verbose”. If TRUE, progress reports are printed during calculation.

Details

This computes four standard summary statistics for a point pattern: the empty space function $F(r)$, nearest neighbour distance distribution function $G(r)$, van Lieshout-Baddeley function $J(r)$ and Ripley's function $K(r)$. The real work is done by `Fest`, `Gest`, `Jest` and `Kest` respectively. Consult the help files for these functions for further information about the statistical interpretation of F, G, J and K.

If `verb` is `TRUE`, then "progress reports" (just indications of completion) are printed out when the calculations are finished for each of the four function types.

The overall title of the array of four functions (for plotting by `plot.fasp`) will be formed from the argument `dataname`. If this is not given, it defaults to the expression for `pp` given in the call to `allstats`.

Value

A list of length 4 containing the F, G, J and K functions respectively.

The list can be plotted directly using `plot` (which dispatches to `plot.solist`).

Each list entry retains the format of the output of the relevant estimating routine `Fest`, `Gest`, `Jest` or `Kest`. Thus each entry in the list is a function value table (object of class `"fv"`, see `fv.object`).

The default formulae for plotting these functions are `cbind(km,theo) ~ r` for F, G, and J, and `cbind(trans,theo) ~ r` for K.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`plot.solist`, `plot.fv`, `fv.object`, `Fest`, `Gest`, `Jest`, `Kest`

Examples

```
data(swedishpines)
        a <- allstats(swedishpines,dataname="Swedish Pines")
        ## Not run:
        plot(a)
        plot(a, subset=list("r<=15","r<=15","r<=15","r<=50"))

## End(Not run)
```

`alltypes` *Calculate Summary Statistic for All Types in a Multitype Point Pattern*

Description

Given a marked point pattern, this computes the estimates of a selected summary function (F,G, J, K etc) of the pattern, for all possible combinations of marks, and returns these functions in an array.

Usage

```
alltypes(X, fun="K", ...,
         dataname=NULL,verb=FALSE,envelope=FALSE,reuse=TRUE)
```

Arguments

`X`	The observed point pattern, for which summary function estimates are required. An object of class `"ppp"` or `"lpp"`.
`fun`	The summary function. Either an R function, or a character string indicating the summary function required. Options for strings are `"F"`, `"G"`, `"J"`, `"K"`, `"L"`, `"pcf"`, `"Gcross"`, `"Jcross"`, `"Kcross"`, `"Lcross"`, `"Gdot"`, `"Jdot"`, `"Kdot"`, `"Ldot"`.
`...`	Arguments passed to the summary function (and to the function `envelope` if appropriate)
`dataname`	Character string giving an optional (alternative) name to the point pattern, different from what is given in the call. This name, if supplied, may be used by `plot.fasp()` in forming the title of the plot. If not supplied it defaults to the parsing of the argument supplied as `X` in the call.
`verb`	Logical value. If `verb` is true then terse "progress reports" (just the values of the mark indices) are printed out when the calculations for that combination of marks are completed.
`envelope`	Logical value. If `envelope` is true, then simulation envelopes of the summary function will also be computed. See Details.
`reuse`	Logical value indicating whether the envelopes in each panel should be based on the same set of simulated patterns (`reuse=TRUE`) or on different, independent sets of simulated patterns (`reuse=FALSE`).

Details

This routine is a convenient way to analyse the dependence between types in a multitype point pattern. It computes the estimates of a selected summary function of the pattern, for all possible combinations of marks. It returns these functions in an array (an object of class `"fasp"`) amenable to plotting by `plot.fasp()`.

The argument `fun` specifies the summary function that will be evaluated for each type of point, or for each pair of types. It may be either an R function or a character string.

Suppose that the points have possible types $1, 2, \ldots, m$ and let X_i denote the pattern of points of type i only.

If `fun="F"` then this routine calculates, for each possible type i, an estimate of the Empty Space Function $F_i(r)$ of X_i. See `Fest` for explanation of the empty space function. The estimate is computed by applying `Fest` to X_i with the optional arguments `...`.

If `fun` is `"Gcross"`, `"Jcross"`, `"Kcross"` or `"Lcross"`, the routine calculates, for each pair of types (i, j), an estimate of the "`i`-to`j`" cross-type function $G_{ij}(r)$, $J_{ij}(r)$, $K_{ij}(r)$ or $L_{ij}(r)$ respectively describing the dependence between X_i and X_j. See `Gcross`, `Jcross`, `Kcross` or `Lcross` respectively for explanation of these functions. The estimate is computed by applying the relevant function (`Gcross` etc) to `X` using each possible value of the arguments `i,j`, together with the optional arguments `...`.

If `fun` is `"pcf"` the routine calculates the cross-type pair correlation function `pcfcross` between each pair of types.

If fun is "Gdot", "Jdot", "Kdot" or "Ldot", the routine calculates, for each type i, an estimate of the "i-to-any" dot-type function $G_{i\bullet}(r)$, $J_{i\bullet}(r)$ or $K_{i\bullet}(r)$ or $L_{i\bullet}(r)$ respectively describing the dependence between X_i and X. See Gdot, Jdot, Kdot or Ldot respectively for explanation of these functions. The estimate is computed by applying the relevant function (Gdot etc) to X using each possible value of the argument i, together with the optional arguments ....

The letters "G", "J", "K" and "L" are interpreted as abbreviations for Gcross, Jcross, Kcross and Lcross respectively, assuming the point pattern is marked. If the point pattern is unmarked, the appropriate function Fest, Jest, Kest or Lest is invoked instead.

If envelope=TRUE, then as well as computing the value of the summary function for each combination of types, the algorithm also computes simulation envelopes of the summary function for each combination of types. The arguments ... are passed to the function envelope to control the number of simulations, the random process generating the simulations, the construction of envelopes, and so on.

Value

A function array (an object of class "fasp", see fasp.object). This can be plotted using plot.fasp.

If the pattern is not marked, the resulting "array" has dimensions 1×1. Otherwise the following is true:

If fun="F", the function array has dimensions $m \times 1$ where m is the number of different marks in the point pattern. The entry at position [i,1] in this array is the result of applying Fest to the points of type i only.

If fun is "Gdot", "Jdot", "Kdot" or "Ldot", the function array again has dimensions $m \times 1$. The entry at position [i,1] in this array is the result of Gdot(X, i), Jdot(X, i) Kdot(X, i) or Ldot(X, i) respectively.

If fun is "Gcross", "Jcross", "Kcross" or "Lcross" (or their abbreviations "G", "J", "K" or "L"), the function array has dimensions $m \times m$. The [i,j] entry of the function array (for $i \neq j$) is the result of applying the function Gcross, Jcross, Kcross orLcross to the pair of types (i,j). The diagonal [i,i] entry of the function array is the result of applying the univariate function Gest, Jest, Kest or Lest to the points of type i only.

If envelope=FALSE, then each function entry fns[[i]] retains the format of the output of the relevant estimating routine Fest, Gest, Jest, Kest, Lest, Gcross, Jcross ,Kcross, Lcross, Gdot, Jdot, Kdot or Ldot The default formulae for plotting these functions are cbind(km,theo) ~ r for F, G, and J functions, and cbind(trans,theo) ~ r for K and L functions.

If envelope=TRUE, then each function entry fns[[i]] has the same format as the output of the envelope command.

Note

Sizeable amounts of memory may be needed during the calculation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

See Also

plot.fasp, fasp.object, Fest, Gest, Jest, Kest, Lest, Gcross, Jcross, Kcross, Lcross, Gdot, Jdot, Kdot, envelope.

Examples

```
   # bramblecanes (3 marks).
   bram <- bramblecanes

   bF <- alltypes(bram,"F",verb=TRUE)
   plot(bF)
   if(interactive()) {
     plot(alltypes(bram,"G"))
     plot(alltypes(bram,"Gdot"))
   }

   # Swedishpines (unmarked).
  swed <- swedishpines

   plot(alltypes(swed,"K"))

   plot(alltypes(amacrine, "pcf"), ylim=c(0,1.3))

   # A setting where you might REALLY want to use dataname:
   ## Not run:
   xxx <- alltypes(ppp(Melvin$x,Melvin$y,
                window=as.owin(c(5,20,15,50)),marks=clyde),
                fun="F",verb=TRUE,dataname="Melvin")

## End(Not run)

   # envelopes
   bKE <- alltypes(bram,"K",envelope=TRUE,nsim=19)
   ## Not run:
   bFE <- alltypes(bram,"F",envelope=TRUE,nsim=19,global=TRUE)

## End(Not run)

   # extract one entry
   as.fv(bKE[1,1])
```

---

angles.psp — *Orientation Angles of Line Segments*

Description

Computes the orientation angle of each line segment in a line segment pattern.

Usage

```
angles.psp(x, directed=FALSE)
```

Arguments

x	A line segment pattern (object of class "psp").
directed	Logical flag. See details.

Details

For each line segment, the angle of inclination to the x-axis (in radians) is computed, and the angles are returned as a numeric vector.

If `directed=TRUE`, the directed angle of orientation is computed. The angle respects the sense of direction from `(x0,y0)` to `(x1,y1)`. The values returned are angles in the full range from $-\pi$ to π. The angle is computed as `atan2(y1-y0,x1-x0)`. See `atan2`.

If `directed=FALSE`, the undirected angle of orientation is computed. Angles differing by π are regarded as equivalent. The values returned are angles in the range from 0 to π. These angles are computed by first computing the directed angle, then adding π to any negative angles.

Value

Numeric vector.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`summary.psp`, `midpoints.psp`, `lengths.psp`

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
b <- angles.psp(a)
```

`anova.lppm` *ANOVA for Fitted Point Process Models on Linear Network*

Description

Performs analysis of deviance for two or more fitted point process models on a linear network.

Usage

```
## S3 method for class 'lppm'
anova(object, ..., test=NULL)
```

Arguments

`object`	A fitted point process model on a linear network (object of class `"lppm"`).
`...`	One or more fitted point process models on the same linear network.
`test`	Character string, partially matching one of `"Chisq"`, `"F"` or `"Cp"`.

Details

This is a method for `anova` for fitted point process models on a linear network (objects of class `"lppm"`, usually generated by the model-fitting function `lppm`).

If the fitted models are all Poisson point processes, then this function performs an Analysis of Deviance of the fitted models. The output shows the deviance differences (i.e. 2 times log likelihood ratio), the difference in degrees of freedom, and (if `test="Chi"`) the two-sided p-values for the chi-squared tests. Their interpretation is very similar to that in `anova.glm`.

If some of the fitted models are *not* Poisson point processes, then the deviance difference is replaced by the adjusted composite likelihood ratio (Pace et al, 2011; Baddeley et al, 2014).

Value

An object of class `"anova"`, or `NULL`.

Errors and warnings

models not nested: There may be an error message that the models are not "nested". For an Analysis of Deviance the models must be nested, i.e. one model must be a special case of the other. For example the point process model with formula `~x` is a special case of the model with formula `~x+y`, so these models are nested. However the two point process models with formulae `~x` and `~y` are not nested.

If you get this error message and you believe that the models should be nested, the problem may be the inability of R to recognise that the two formulae are nested. Try modifying the formulae to make their relationship more obvious.

different sizes of dataset: There may be an error message from `anova.glmlist` that "models were not all fitted to the same size of dataset". This generally occurs when the point process models are fitted on different linear networks.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Ang, Q.W. (2010) *Statistical methodology for events on a network*. Master's thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

Baddeley, A., Turner, R. and Rubak, E. (2015) Adjusted composite likelihood ratio test for Gibbs point processes. *Journal of Statistical Computation and Simulation* **86** (5) 922–941. DOI: 10.1080/00949655.2015.10445

McSwiggan, G., Nair, M.G. and Baddeley, A. (2012) Fitting Poisson point process models to events on a linear network. Manuscript in preparation.

Pace, L., Salvan, A. and Sartori, N. (2011) Adjusting composite likelihood ratio statistics. *Statistica Sinica* **21**, 129–148.

See Also

`lppm`

Examples

```
X <- runiflpp(10, simplenet)
mod0 <- lppm(X ~1)
modx <- lppm(X ~x)
anova(mod0, modx, test="Chi")
```

---

anova.mppm *ANOVA for Fitted Point Process Models for Replicated Patterns*

Description

Performs analysis of deviance for one or more point process models fitted to replicated point pattern data.

Usage

```
## S3 method for class 'mppm'
anova(object, ...,
                    test=NULL, adjust=TRUE,
                    fine=FALSE, warn=TRUE)
```

Arguments

`object`	Object of class "mppm" representing a point process model that was fitted to replicated point patterns.
`...`	Optional. Additional objects of class "mppm".
`test`	Type of hypothesis test to perform. A character string, partially matching one of "Chisq", "LRT", "Rao", "score", "F" or "Cp", or NULL indicating that no test should be performed.
`adjust`	Logical value indicating whether to correct the pseudolikelihood ratio when some of the models are not Poisson processes.
`fine`	Logical value passed to `vcov.ppm` indicating whether to use a quick estimate (`fine=FALSE`, the default) or a slower, more accurate estimate (`fine=TRUE`) of the variance of the fitted coefficients of each model. Relevant only when some of the models are not Poisson and `adjust=TRUE`.
`warn`	Logical value indicating whether to issue warnings if problems arise.

Details

This is a method for `anova` for comparing several fitted point process models of class "mppm", usually generated by the model-fitting function `mppm`).

If the fitted models are all Poisson point processes, then this function performs an Analysis of Deviance of the fitted models. The output shows the deviance differences (i.e. 2 times log likelihood ratio), the difference in degrees of freedom, and (if `test="Chi"`) the two-sided p-values for the chi-squared tests. Their interpretation is very similar to that in `anova.glm`.

If some of the fitted models are *not* Poisson point processes, the 'deviance' differences in this table are 'pseudo-deviances' equal to 2 times the differences in the maximised values of the log pseudolikelihood (see `ppm`). It is not valid to compare these values to the chi-squared distribution. In this case, if `adjust=TRUE` (the default), the pseudo-deviances will be adjusted using the method of Pace et al (2011) and Baddeley, Turner and Rubak (2015) so that the chi-squared test is valid. It is strongly advisable to perform this adjustment.

The argument `test` determines which hypothesis test, if any, will be performed to compare the models. The argument `test` should be a character string, partially matching one of `"Chisq"`, `"F"` or `"Cp"`, or `NULL`. The first option `"Chisq"` gives the likelihood ratio test based on the asymptotic chi-squared distribution of the deviance difference. The meaning of the other options is explained in `anova.glm`. For random effects models, only `"Chisq"` is available, and again gives the likelihood ratio test.

Value

An object of class `"anova"`, or `NULL`.

Error messages

An error message that reports *system is computationally singular* indicates that the determinant of the Fisher information matrix of one of the models was either too large or too small for reliable numerical calculation. See `vcov.ppm` for suggestions on how to handle this.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley `<Adrian.Baddeley@curti`
Rolf Turner `<r.turner@auckland.ac.nz>` and Ege Rubak `<rubak@math.aau.dk>`.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

Baddeley, A., Turner, R. and Rubak, E. (2015) Adjusted composite likelihood ratio test for Gibbs point processes. *Journal of Statistical Computation and Simulation* **86** (5) 922–941. DOI: 10.1080/00949655.2015.10445

Pace, L., Salvan, A. and Sartori, N. (2011) Adjusting composite likelihood ratio statistics. *Statistica Sinica* **21**, 129–148.

See Also

`mppm`

Examples

```
H <- hyperframe(X=waterstriders)
mod0 <- mppm(X~1, data=H, Poisson())
modx <- mppm(X~x, data=H, Poisson())
anova(mod0, modx, test="Chi")

mod0S <- mppm(X~1, data=H, Strauss(2))
modxS <- mppm(X~x, data=H, Strauss(2))
anova(mod0S, modxS, test="Chi")
```

anova.ppm *ANOVA for Fitted Point Process Models*

Description

Performs analysis of deviance for one or more fitted point process models.

Usage

```
  ## S3 method for class 'ppm'
anova(object, ..., test=NULL,
                     adjust=TRUE, warn=TRUE, fine=FALSE)
```

Arguments

`object`	A fitted point process model (object of class "ppm").
`...`	Optional. Additional objects of class "ppm".
`test`	Character string, partially matching one of "Chisq", "LRT", "Rao", "score", "F" or "Cp", or NULL indicating that no test should be performed.
`adjust`	Logical value indicating whether to correct the pseudolikelihood ratio when some of the models are not Poisson processes.
`warn`	Logical value indicating whether to issue warnings if problems arise.
`fine`	Logical value, passed to `vcov.ppm`, indicating whether to use a quick estimate (`fine=FALSE`, the default) or a slower, more accurate estimate (`fine=TRUE`) of variance terms. Relevant only when some of the models are not Poisson and `adjust=TRUE`.

Details

This is a method for `anova` for fitted point process models (objects of class "ppm", usually generated by the model-fitting function `ppm`).

If the fitted models are all Poisson point processes, then by default, this function performs an Analysis of Deviance of the fitted models. The output shows the deviance differences (i.e. 2 times log likelihood ratio), the difference in degrees of freedom, and (if `test="Chi"` or `test="LRT"`) the two-sided p-values for the chi-squared tests. Their interpretation is very similar to that in `anova.glm`. If `test="Rao"` or `test="score"`, the *score test* (Rao, 1948) is performed instead.

If some of the fitted models are *not* Poisson point processes, the 'deviance' differences in this table are 'pseudo-deviances' equal to 2 times the differences in the maximised values of the log pseudolikelihood (see `ppm`). It is not valid to compare these values to the chi-squared distribution. In this case, if `adjust=TRUE` (the default), the pseudo-deviances will be adjusted using the method of Pace et al (2011) and Baddeley et al (2015) so that the chi-squared test is valid. It is strongly advisable to perform this adjustment.

Value

An object of class "anova", or NULL.

Errors and warnings

models not nested: There may be an error message that the models are not "nested". For an Analysis of Deviance the models must be nested, i.e. one model must be a special case of the other. For example the point process model with formula `~x` is a special case of the model with formula `~x+y`, so these models are nested. However the two point process models with formulae `~x` and `~y` are not nested.

If you get this error message and you believe that the models should be nested, the problem may be the inability of R to recognise that the two formulae are nested. Try modifying the formulae to make their relationship more obvious.

different sizes of dataset: There may be an error message from `anova.glmlist` that "models were not all fitted to the same size of dataset". This implies that the models were fitted using different quadrature schemes (see `quadscheme`) and/or with different edge corrections or different values of the border edge correction distance `rbord`.

To ensure that models are comparable, check the following:

- the models must all have been fitted to the same point pattern dataset, in the same window.
- all models must have been fitted by the same fitting method as specified by the argument `method` in `ppm`.
- If some of the models depend on covariates, then they should all have been fitted using the same list of covariates, and using `allcovar=TRUE` to ensure that the same quadrature scheme is used.
- all models must have been fitted using the same edge correction as specified by the arguments `correction` and `rbord`. If you did not specify the value of `rbord`, then it may have taken a different value for different models. The default value of `rbord` is equal to zero for a Poisson model, and otherwise equals the reach (interaction distance) of the interaction term (see `reach`). To ensure that the models are comparable, set `rbord` to equal the maximum reach of the interactions that you are fitting.

Error messages

An error message that reports *system is computationally singular* indicates that the determinant of the Fisher information matrix of one of the models was either too large or too small for reliable numerical calculation. See `vcov.ppm` for suggestions on how to handle this.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Turner, R. and Rubak, E. (2015) Adjusted composite likelihood ratio test for Gibbs point processes. *Journal of Statistical Computation and Simulation* **86** (5) 922–941. DOI: 10.1080/00949655.2015.10445

Pace, L., Salvan, A. and Sartori, N. (2011) Adjusting composite likelihood ratio statistics. *Statistica Sinica* **21**, 129–148.

Rao, C.R. (1948) Large sample tests of statistical hypotheses concerning several parameters with applications to problems of estimation. *Proceedings of the Cambridge Philosophical Society* **44**, 50–57.

See Also

`ppm`, `vcov.ppm`

Examples

```
mod0 <- ppm(swedishpines ~1)
modx <- ppm(swedishpines ~x)
# Likelihood ratio test
anova(mod0, modx, test="Chi")
# Score test
anova(mod0, modx, test="Rao")

# Single argument
modxy <- ppm(swedishpines ~x + y)
anova(modxy, test="Chi")

# Adjusted composite likelihood ratio test
modP <- ppm(swedishpines ~1, rbord=9)
modS <- ppm(swedishpines ~1, Strauss(9))
anova(modP, modS, test="Chi")
```

---

anova.slrm — *Analysis of Deviance for Spatial Logistic Regression Models*

Description

Performs Analysis of Deviance for two or more fitted Spatial Logistic Regression models.

Usage

```
  ## S3 method for class 'slrm'
anova(object, ..., test = NULL)
```

Arguments

`object`	a fitted spatial logistic regression model. An object of class `"slrm"`.
`...`	additional objects of the same type (optional).
`test`	a character string, (partially) matching one of `"Chisq"`, `"F"` or `"Cp"`, indicating the reference distribution that should be used to compute p-values.

Details

This is a method for `anova` for fitted spatial logistic regression models (objects of class `"slrm"`, usually obtained from the function `slrm`).

The output shows the deviance differences (i.e. 2 times log likelihood ratio), the difference in degrees of freedom, and (if `test="Chi"`) the two-sided p-values for the chi-squared tests. Their interpretation is very similar to that in `anova.glm`.

Value

An object of class `"anova"`, inheriting from class `"data.frame"`, representing the analysis of deviance table.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

slrm

Examples

```
X <- rpoispp(42)
fit0 <- slrm(X ~ 1)
fit1 <- slrm(X ~ x+y)
anova(fit0, fit1, test="Chi")
```

---

anylist *List of Objects*

Description

Make a list of objects of any type.

Usage

```
anylist(...)
as.anylist(x)
```

Arguments

...	Any number of arguments of any type.
x	A list.

Details

An object of class "anylist" is a list of objects that the user intends to treat in a similar fashion.

For example it may be desired to plot each of the objects side-by-side: this can be done using the function plot.anylist.

The objects can belong to any class; they may or may not all belong to the same class.

In the **spatstat** package, various functions produce an object of class "anylist".

Value

A list, belonging to the class "anylist", containing the original objects.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

solist, as.solist, anylapply.

Examples

```
anylist(cells, intensity(cells), Kest(cells))
```

---

anyNA.im *Check Whether Image Contains NA Values*

Description

Checks whether any pixel values in a pixel image are NA (meaning that the pixel lies outside the domain of definition of the image).

Usage

```
  ## S3 method for class 'im'
anyNA(x, recursive = FALSE)
```

Arguments

x	A pixel image (object of class "im").
recursive	Ignored.

Details

The function anyNA is generic: anyNA(x) is a faster alternative to any(is.na(x)).

This function anyNA.im is a method for the generic anyNA defined for pixel images. It returns the value TRUE if any of the pixel values in x are NA, and and otherwise returns FALSE.

Value

A single logical value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

im.object

Examples

```
anyNA(as.im(letterR))
```

append.psp *Combine Two Line Segment Patterns*

Description

Combine two line segment patterns into a single pattern.

Usage

```
append.psp(A, B)
```

Arguments

A,B	Line segment patterns (objects of class "psp").

Details

This function is used to superimpose two line segment patterns A and B.

The two patterns must have **identical** windows. If one pattern has marks, then the other must also have marks of the same type. It the marks are data frames then the number of columns of these data frames, and the names of the columns must be identical.

(To combine two point patterns, see superimpose).

Value

Another line segment pattern (object of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

psp, as.psp, superimpose,

Examples

```
X <- psp(runif(20), runif(20), runif(20), runif(20),  window=owin())
Y <- psp(runif(5), runif(5), runif(5), runif(5),  window=owin())
append.psp(X,Y)
```

applynbd *Apply Function to Every Neighbourhood in a Point Pattern*

Description

Visit each point in a point pattern, find the neighbouring points, and apply a given function to them.

Usage

```
applynbd(X, FUN, N=NULL, R=NULL, criterion=NULL, exclude=FALSE, ...)
```

Arguments

X	Point pattern. An object of class "ppp", or data which can be converted into this format by as.ppp.
FUN	Function to be applied to each neighbourhood. The arguments of FUN are described under **Details**.
N	Integer. If this argument is present, the neighbourhood of a point of X is defined to consist of the N points of X which are closest to it.
R	Nonnegative numeric value. If this argument is present, the neighbourhood of a point of X is defined to consist of all points of X which lie within a distance R of it.
criterion	Function. If this argument is present, the neighbourhood of a point of X is determined by evaluating this function. See under **Details**.
exclude	Logical. If TRUE then the point currently being visited is excluded from its own neighbourhood.
...	extra arguments passed to the function FUN. They must be given in the form name=value.

Details

This is an analogue of apply for point patterns. It visits each point in the point pattern X, determines which points of X are "neighbours" of the current point, applies the function FUN to this neighbourhood, and collects the values returned by FUN.

The definition of "neighbours" depends on the arguments N, R and criterion. Also the argument exclude determines whether the current point is excluded from its own neighbourhood.

- If N is given, then the neighbours of the current point are the N points of X which are closest to the current point (including the current point itself unless exclude=TRUE).
- If R is given, then the neighbourhood of the current point consists of all points of X which lie closer than a distance R from the current point.
- If criterion is given, then it must be a function with two arguments dist and drank which will be vectors of equal length. The interpretation is that dist[i] will be the distance of a point from the current point, and drank[i] will be the rank of that distance (the three points closest to the current point will have rank 1, 2 and 3). This function must return a logical vector of the same length as dist and drank whose i-th entry is TRUE if the corresponding point should be included in the neighbourhood. See the examples below.

- If more than one of the arguments `N`, `R` and `criterion` is given, the neighbourhood is defined as the *intersection* of the neighbourhoods specified by these arguments. For example if `N=3` and `R=5` then the neighbourhood is formed by finding the 3 nearest neighbours of current point, and retaining only those neighbours which lie closer than 5 units from the current point.

When `applynbd` is executed, each point of `X` is visited, and the following happens for each point:

- the neighbourhood of the current point is determined according to the chosen rule, and stored as a point pattern `Y`;
- the function `FUN` is called as:

  `FUN(Y=Y, current=current, dists=dists, dranks=dranks, ...)`

  where `current` is the location of the current point (in a format explained below), `dists` is a vector of distances from the current point to each of the points in `Y`, `dranks` is a vector of the ranks of these distances with respect to the full point pattern `X`, and `...` are the arguments passed from the call to `applynbd`;
- The result of the call to `FUN` is stored.

The results of each call to `FUN` are collected and returned according to the usual rules for `apply` and its relatives. See the **Value** section of this help file.

The format of the argument `current` is as follows. If `X` is an unmarked point pattern, then `current` is a vector of length 2 containing the coordinates of the current point. If `X` is marked, then `current` is a point pattern containing exactly one point, so that `current$x` is its x-coordinate and `current$marks` is its mark value. In either case, the coordinates of the current point can be referred to as `current$x` and `current$y`.

Note that `FUN` will be called exactly as described above, with each argument named explicitly. Care is required when writing the function `FUN` to ensure that the arguments will match up. See the Examples.

See `markstat` for a common use of this function.

To simply tabulate the marks in every `R`-neighbourhood, use `marktable`.

Value

Similar to the result of `apply`. If each call to `FUN` returns a single numeric value, the result is a vector of dimension `npoints(X)`, the number of points in `X`. If each call to `FUN` returns a vector of the same length `m`, then the result is a matrix of dimensions `c(m,n)`; note the transposition of the indices, as usual for the family of `apply` functions. If the calls to `FUN` return vectors of different lengths, the result is a list of length `npoints(X)`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`ppp.object`, `apply`, `markstat`, `marktable`

Examples

```
redwood
# count the number of points within radius 0.2 of each point of X
nneighbours <- applynbd(redwood, R=0.2, function(Y, ...){npoints(Y)-1})
```

```
# equivalent to:
nneighbours <- applynbd(redwood, R=0.2, function(Y, ...){npoints(Y)}, exclude=TRUE)

# compute the distance to the second nearest neighbour of each point
secondnndist <- applynbd(redwood, N = 2,
                         function(dists, ...){max(dists)},
                         exclude=TRUE)

# marked point pattern
trees <- longleaf

# compute the median of the marks of all neighbours of a point
# (see also 'markstat')
dbh.med <- applynbd(trees, R=90, exclude=TRUE,
               function(Y, ...) { median(marks(Y))})


# ANIMATION explaining the definition of the K function
# (arguments `fullpicture' and 'rad' are passed to FUN)

if(interactive()) {
showoffK <- function(Y, current, dists, dranks, fullpicture,rad) {
plot(fullpicture, main="")
points(Y, cex=2)
      ux <- current[["x"]]
      uy <- current[["y"]]
points(ux, uy, pch="+",cex=3)
theta <- seq(0,2*pi,length=100)
polygon(ux + rad * cos(theta), uy+rad*sin(theta))
text(ux + rad/3, uy + rad/2,npoints(Y),cex=3)
if(interactive()) Sys.sleep(if(runif(1) < 0.1) 1.5 else 0.3)
return(npoints(Y))
}
applynbd(redwood, R=0.2, showoffK, fullpicture=redwood, rad=0.2, exclude=TRUE)

# animation explaining the definition of the G function

showoffG <- function(Y, current, dists, dranks, fullpicture) {
plot(fullpicture, main="")
points(Y, cex=2)
      u <- current
points(u[1],u[2],pch="+",cex=3)
v <- c(Y$x[1],Y$y[1])
segments(u[1],u[2],v[1],v[2],lwd=2)
w <- (u + v)/2
nnd <- dists[1]
text(w[1],w[2],round(nnd,3),cex=2)
if(interactive()) Sys.sleep(if(runif(1) < 0.1) 1.5 else 0.3)
return(nnd)
}

applynbd(cells, N=1, showoffG, exclude=TRUE, fullpicture=cells)
}
```

---

area.owin *Area of a Window*

Description

Computes the area of a window

Usage

```
 area(w)

 ## S3 method for class 'owin'
area(w)

 ## Default S3 method:
area(w)

 ## S3 method for class 'owin'
volume(x)
```

Arguments

w	A window, whose area will be computed. This should be an object of class owin, or can be given in any format acceptable to as.owin().
x	Object of class owin

Details

If the window w is of type "rectangle" or "polygonal", the area of this rectangular window is computed by analytic geometry. If w is of type "mask" the area of the discrete raster approximation of the window is computed by summing the binary image values and adjusting for pixel size.

The function volume.owin is identical to area.owin except for the argument name. It is a method for the generic function volume.

Value

A numerical value giving the area of the window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

perimeter, diameter.owin, owin.object, as.owin

Examples

```
 w <- unit.square()
 area(w)
      # returns 1.00000

 k <- 6
 theta <- 2 * pi * (0:(k-1))/k
 co <- cos(theta)
 si <- sin(theta)
```

```
mas <- owin(c(-1,1), c(-1,1), poly=list(x=co, y=si))
area(mas)
    # returns approx area of k-gon

mas <- as.mask(square(2), eps=0.01)
X <- raster.x(mas)
Y <- raster.y(mas)
mas$m <- ((X - 1)^2 + (Y - 1)^2 <= 1)
area(mas)
     # returns 3.14 approx
```

---

areaGain *Difference of Disc Areas*

Description

Computes the area of that part of a disc that is not covered by other discs.

Usage

```
areaGain(u, X, r, ..., W=as.owin(X), exact=FALSE,
                     ngrid=spatstat.options("ngrid.disc"))
```

Arguments

u	Coordinates of the centre of the disc of interest. A vector of length 2. Alternatively, a point pattern (object of class "ppp").
X	Locations of the centres of other discs. A point pattern (object of class "ppp").
r	Disc radius, or vector of disc radii.
...	Arguments passed to distmap to determine the pixel resolution, when exact=FALSE.
W	Window (object of class "owin") in which the area should be computed.
exact	Choice of algorithm. If exact=TRUE, areas are computed exactly using analytic geometry. If exact=FALSE then a faster algorithm is used to compute a discrete approximation to the areas.
ngrid	Integer. Number of points in the square grid used to compute the discrete approximation, when exact=FALSE.

Details

This function computes the area of that part of the disc of radius r centred at the location u that is *not* covered by any of the discs of radius r centred at the points of the pattern X. This area is important in some calculations related to the area-interaction model AreaInter.

If u is a point pattern and r is a vector, the result is a matrix, with one row for each point in u and one column for each entry of r. The [i,j] entry in the matrix is the area of that part of the disc of radius r[j] centred at the location u[i] that is *not* covered by any of the discs of radius r[j] centred at the points of the pattern X.

If W is not NULL, then the areas are computed only inside the window W.

Value

A matrix with one row for each point in u and one column for each value in r.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

AreaInter, areaLoss

Examples

```
data(cells)
u <- c(0.5,0.5)
areaGain(u, cells, 0.1)
```

AreaInter — *The Area Interaction Point Process Model*

Description

Creates an instance of the Area Interaction point process model (Widom-Rowlinson penetrable spheres model) which can then be fitted to point pattern data.

Usage

```
AreaInter(r)
```

Arguments

r	The radius of the discs in the area interaction process

Details

This function defines the interpoint interaction structure of a point process called the Widom-Rowlinson penetrable sphere model or area-interaction process. It can be used to fit this model to point pattern data.

The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the area interaction structure is yielded by the function AreaInter(). See the examples below.

In **standard form**, the area-interaction process (Widom and Rowlinson, 1970; Baddeley and Van Lieshout, 1995) with disc radius r, intensity parameter κ and interaction parameter γ is a point process with probability density

$$f(x_1, \dots, x_n) = \alpha \kappa^{n(x)} \gamma^{-A(x)}$$

for a point pattern x, where $x_1, \dots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, and $A(x)$ is the area of the region formed by the union of discs of radius r centred at the points $x_1, \dots, x_n$. Here α is a normalising constant.

The interaction parameter γ can be any positive number. If $\gamma = 1$ then the model reduces to a Poisson process with intensity κ. If $\gamma < 1$ then the process is regular, while if $\gamma > 1$ the process is clustered. Thus, an area interaction process can be used to model either clustered or regular point patterns. Two points interact if the distance between them is less than $2r$.

The standard form of the model, shown above, is a little complicated to interpret in practical applications. For example, each isolated point of the pattern x contributes a factor $\kappa\gamma^{-\pi r^2}$ to the probability density.

In **spatstat**, the model is parametrised in a different form, which is easier to interpret. In **canonical scale-free form**, the probability density is rewritten as

$$f(x_1, \dots, x_n) = \alpha\beta^{n(x)}\eta^{-C(x)}$$

where β is the new intensity parameter, η is the new interaction parameter, and $C(x) = B(x) - n(x)$ is the interaction potential. Here

$$B(x) = \frac{A(x)}{\pi r^2}$$

is the normalised area (so that the discs have unit area). In this formulation, each isolated point of the pattern contributes a factor β to the probability density (so the first order trend is β). The quantity $C(x)$ is a true interaction potential, in the sense that $C(x) = 0$ if the point pattern x does not contain any points that lie close together (closer than $2r$ units apart).

When a new point u is added to an existing point pattern x, the rescaled potential $-C(x)$ increases by a value between 0 and 1. The increase is zero if u is not close to any point of x. The increase is 1 if the disc of radius r centred at u is completely contained in the union of discs of radius r centred at the data points x_i. Thus, the increase in potential is a measure of how close the new point u is to the existing pattern x. Addition of the point u contributes a factor $\beta\eta^{\delta}$ to the probability density, where δ is the increase in potential.

The old parameters κ, γ of the standard form are related to the new parameters β, η of the canonical scale-free form, by

$$\beta = \kappa\gamma^{-\pi r^2} = \kappa/\eta$$

and

$$\eta = \gamma^{\pi r^2}$$

provided γ and κ are positive and finite.

In the canonical scale-free form, the parameter η can take any nonnegative value. The value $\eta = 1$ again corresponds to a Poisson process, with intensity β. If $\eta < 1$ then the process is regular, while if $\eta > 1$ the process is clustered. The value $\eta = 0$ corresponds to a hard core process with hard core radius r (interaction distance $2r$).

The *nonstationary* area interaction process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location, rather than a constant beta.

Note the only argument of `AreaInter()` is the disc radius `r`. When `r` is fixed, the model becomes an exponential family. The canonical parameters $\log(\beta)$ and $\log(\eta)$ are estimated by `ppm()`, not fixed in `AreaInter()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the area-interaction process with disc radius r.

Warnings

The interaction distance of this process is equal to `2 * r`. Two discs of radius `r` overlap if their centres are closer than `2 * r` units apart.

The estimate of the interaction parameter η is unreliable if the interaction radius `r` is too small or too large. In these situations the model is approximately Poisson so that η is unidentifiable. As a rule of thumb, one can inspect the empty space function of the data, computed by `Fest`. The value $F(r)$ of the empty space function at the interaction radius `r` should be between 0.2 and 0.8.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A.J. and Van Lieshout, M.N.M. (1995). Area-interaction point processes. *Annals of the Institute of Statistical Mathematics* **47** (1995) 601–619.

Widom, B. and Rowlinson, J.S. (1970). New model for the study of liquid-vapor phase transitions. *The Journal of Chemical Physics* **52** (1970) 1670–1684.

See Also

`ppm`, `pairwise.family`, `ppm.object`

`ragsAreaInter` and `rmh` for simulation of area-interaction models.

Examples

```
   # prints a sensible description of itself
   AreaInter(r=0.1)

   # Note the reach is twice the radius
   reach(AreaInter(r=1))

   # Fit the stationary area interaction process to Swedish Pines data
   data(swedishpines)
   ppm(swedishpines, ~1, AreaInter(r=7))

   # Fit the stationary area interaction process to `cells'
   data(cells)
   ppm(cells, ~1, AreaInter(r=0.06))
   # eta=0 indicates hard core process.

   # Fit a nonstationary area interaction with log-cubic polynomial trend
   ## Not run:
   ppm(swedishpines, ~polynom(x/10,y/10,3), AreaInter(r=7))

## End(Not run)
```

areaLoss *Difference of Disc Areas*

Description

Computes the area of that part of a disc that is not covered by other discs.

Usage

```
areaLoss(X, r, ..., W=as.owin(X), subset=NULL,
                exact=FALSE,
                ngrid=spatstat.options("ngrid.disc"))
```

Arguments

X	Locations of the centres of discs. A point pattern (object of class "ppp").
r	Disc radius, or vector of disc radii.
...	Ignored.
W	Optional. Window (object of class "owin") inside which the area should be calculated.
subset	Optional. Index identifying a subset of the points of X for which the area difference should be computed.
exact	Choice of algorithm. If exact=TRUE, areas are computed exactly using analytic geometry. If exact=FALSE then a faster algorithm is used to compute a discrete approximation to the areas.
ngrid	Integer. Number of points in the square grid used to compute the discrete approximation, when exact=FALSE.

Details

This function computes, for each point X[i] in X and for each radius r, the area of that part of the disc of radius r centred at the location X[i] that is *not* covered by any of the other discs of radius r centred at the points X[j] for j not equal to i. This area is important in some calculations related to the area-interaction model AreaInter.

The result is a matrix, with one row for each point in X and one column for each entry of r.

Value

A matrix with one row for each point in X (or X[subset]) and one column for each value in r.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

AreaInter, areaGain, dilated.areas

Examples

```
data(cells)
areaLoss(cells, 0.1)
```

---

as.box3 *Convert Data to Three-Dimensional Box*

Description

Interprets data as the dimensions of a three-dimensional box.

Usage

```
as.box3(...)
```

Arguments

...	Data that can be interpreted as giving the dimensions of a three-dimensional box. See Details.

Details

This function converts data in various formats to an object of class "box3" representing a three-dimensional box (see box3). The arguments ... may be

- an object of class "box3"
- arguments acceptable to box3
- a numeric vector of length 6, interpreted as c(xrange[1],xrange[2],yrange[1],yrange[2],zrange[1],zrange
- an object of class "pp3" representing a three-dimensional point pattern contained in a box.

Value

Object of class "box3".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

box3, pp3

Examples

```
X <- c(0,10,0,10,0,5)
as.box3(X)
X <- pp3(runif(42),runif(42),runif(42), box3(c(0,1)))
as.box3(X)
```

as.boxx *Convert Data to Multi-Dimensional Box*

Description

Interprets data as the dimensions of a multi-dimensional box.

Usage

```
as.boxx(..., warn.owin = TRUE)
```

Arguments

...	Data that can be interpreted as giving the dimensions of a multi-dimensional box. See Details.
warn.owin	Logical value indicating whether to print a warning if a non-rectangular window (object of class "owin") is supplied.

Details

Either a single argument should be provided which is one of the following:

- an object of class "boxx"
- an object of class "box3"
- an object of class "owin"
- a numeric vector of even length, specifying the corners of the box. See Examples

or a list of arguments acceptable to boxx.

Value

A "boxx" object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
# Convert unit square to two dimensional box.
W <- owin()
as.boxx(W)
# Make three dimensional box [0,1]x[0,1]x[0,1] from numeric vector
as.boxx(c(0,1,0,1,0,1))
```

`as.data.frame.envelope` *Coerce Envelope to Data Frame*

Description

Converts an envelope object to a data frame.

Usage

```
## S3 method for class 'envelope'
as.data.frame(x, ..., simfuns=FALSE)
```

Arguments

`x`	Envelope object (class `"envelope"`).
`...`	Ignored.
`simfuns`	Logical value indicating whether the result should include the values of the simulated functions that were used to build the envelope.

Details

This is a method for the generic function `as.data.frame` for the class of envelopes (see `envelope`.

The result is a data frame with columns containing the values of the function argument (usually named `r`), the function estimate for the original point pattern data (`obs`), the upper and lower envelope limits (`hi` and `lo`), and possibly additional columns.

If `simfuns=TRUE`, the result also includes columns of values of the simulated functions that were used to compute the envelope. This is possible only when the envelope was computed with the argument `savefuns=TRUE` in the call to `envelope`.

Value

A data frame.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

Examples

```
E <- envelope(cells, nsim=5, savefuns=TRUE)
tail(as.data.frame(E))
tail(as.data.frame(E, simfuns=TRUE))
```

as.data.frame.hyperframe *Coerce Hyperframe to Data Frame*

Description

Converts a hyperframe to a data frame.

Usage

```
## S3 method for class 'hyperframe'
as.data.frame(x, row.names = NULL,
                                    optional = FALSE, ...,
                                    discard=TRUE, warn=TRUE)
```

Arguments

x	Hyperframe (object of class "hyperframe").
row.names	Optional character vector of row names.
optional	Argument passed to as.data.frame controlling what happens to row names.
...	Ignored.
discard	Logical. Whether to discard columns of the hyperframe that do not contain atomic data. See Details.
warn	Logical. Whether to issue a warning when columns are discarded.

Details

This is a method for the generic function as.data.frame for the class of hyperframes (see hyperframe.

If discard=TRUE, any columns of the hyperframe that do not contain atomic data will be removed (and a warning will be issued if warn=TRUE). If discard=FALSE, then such columns are converted to strings indicating what class of data they originally contained.

Value

A data frame.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
h <- hyperframe(X=1:3, Y=letters[1:3], f=list(sin, cos, tan))
as.data.frame(h, discard=TRUE, warn=FALSE)
as.data.frame(h, discard=FALSE)
```

`as.data.frame.im` *Convert Pixel Image to Data Frame*

Description

Convert a pixel image to a data frame

Usage

```
  ## S3 method for class 'im'
as.data.frame(x, ...)
```

Arguments

`x`	A pixel image (object of class `"im"`).
`...`	Further arguments passed to `as.data.frame.default` to determine the row names and other features.

Details

This function takes the pixel image `x` and returns a data frame with three columns containing the pixel coordinates and the pixel values.

The data frame entries are automatically sorted in increasing order of the `x` coordinate (and in increasing order of `y` within `x`).

Value

A data frame.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

Examples

```
# artificial image
Z <- setcov(square(1))

Y <- as.data.frame(Z)

head(Y)
```

`as.data.frame.owin` *Convert Window to Data Frame*

Description

Converts a window object to a data frame.

Usage

```
## S3 method for class 'owin'
as.data.frame(x, ..., drop=TRUE)
```

Arguments

`x`	Window (object of class `"owin"`).
`...`	Further arguments passed to `as.data.frame.default` to determine the row names and other features.
`drop`	Logical value indicating whether to discard pixels that are outside the window, when `x` is a binary mask.

Details

This function returns a data frame specifying the coordinates of the window.

If `x` is a binary mask window, the result is a data frame with columns `x` and `y` containing the spatial coordinates of each *pixel*. If `drop=TRUE` (the default), only pixels inside the window are retained. If `drop=FALSE`, all pixels are retained, and the data frame has an extra column `inside` containing the logical value of each pixel (`TRUE` for pixels inside the window, `FALSE` for outside).

If `x` is a rectangle or a polygonal window, the result is a data frame with columns `x` and `y` containing the spatial coordinates of the *vertices* of the window. If the boundary consists of several polygons, the data frame has additional columns `id`, identifying which polygon is being traced, and `sign`, indicating whether the polygon is an outer or inner boundary (`sign=1` and `sign=-1` respectively).

Value

A data frame with columns named `x` and `y`, and possibly other columns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`as.data.frame.im`

Examples

```
as.data.frame(square(1))

holey <- owin(poly=list(
                     list(x=c(0,10,0), y=c(0,0,10)),
                     list(x=c(2,2,4,4), y=c(2,4,4,2))))
as.data.frame(holey)
```

---

as.data.frame.ppp *Coerce Point Pattern to a Data Frame*

Description

Extracts the coordinates of the points in a point pattern, and their marks if any, and returns them in a data frame.

Usage

```
## S3 method for class 'ppp'
as.data.frame(x, row.names = NULL, ...)
```

Arguments

x	Point pattern (object of class "ppp").
row.names	Optional character vector of row names.
...	Ignored.

Details

This is a method for the generic function as.data.frame for the class "ppp" of point patterns.

It extracts the coordinates of the points in the point pattern, and returns them as columns named x and y in a data frame. If the points were marked, the marks are returned as a column named marks with the same type as in the point pattern dataset.

Value

A data frame.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
data(amacrine)
df <- as.data.frame(amacrine)
df[1:5,]
```

`as.data.frame.psp` *Coerce Line Segment Pattern to a Data Frame*

Description

Extracts the coordinates of the endpoints in a line segment pattern, and their marks if any, and returns them in a data frame.

Usage

```
## S3 method for class 'psp'
as.data.frame(x, row.names = NULL, ...)
```

Arguments

`x`	Line segment pattern (object of class "psp").
`row.names`	Optional character vector of row names.
`...`	Ignored.

Details

This is a method for the generic function `as.data.frame` for the class "psp" of line segment patterns.

It extracts the coordinates of the endpoints of the line segments, and returns them as columns named `x0`, `y0`, `x1` and `y1` in a data frame. If the line segments were marked, the marks are appended as an extra column or columns to the data frame which is returned. If the marks are a vector then a single column named `marks` is appended. in the data frame, with the same type as in the line segment pattern dataset. If the marks are a data frame, then the columns of this data frame are appended (retaining their names).

Value

A data frame with 4 or 5 columns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
data(copper)
df <- as.data.frame(copper$Lines)
```

as.data.frame.tess *Convert Tessellation to Data Frame*

Description

Converts a spatial tessellation object to a data frame.

Usage

```
## S3 method for class 'tess'
as.data.frame(x, ...)
```

Arguments

`x`	Tessellation (object of class "tess").
`...`	Further arguments passed to `as.data.frame.owin` or `as.data.frame.im` and ultimately to `as.data.frame.default` to determine the row names and other features.

Details

This function converts the tessellation `x` to a data frame.

If `x` is a pixel image tessellation (a pixel image with factor values specifying the tile membership of each pixel) then this pixel image is converted to a data frame by `as.data.frame.im`. The result is a data frame with columns `x` and `y` giving the pixel coordinates, and `Tile` identifying the tile containing the pixel.

If `x` is a tessellation consisting of a rectangular grid of tiles or a list of polygonal tiles, then each tile is converted to a data frame by `as.data.frame.owin`, and these data frames are joined together, yielding a single large data frame containing columns `x`, `y` giving the coordinates of vertices of the polygons, and `Tile` identifying the tile.

Value

A data frame with columns named `x`, `y`, `Tile`, and possibly other columns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`as.data.frame.owin`, `as.data.frame.im`

Examples

```
Z <- as.data.frame(dirichlet(cells))
head(Z, 10)
```

as.function.fv *Convert Function Value Table to Function*

Description

Converts an object of class "fv" to an R language function.

Usage

```
  ## S3 method for class 'fv'
as.function(x, ..., value=".y", extrapolate=FALSE)

  ## S3 method for class 'rhohat'
as.function(x, ..., value=".y", extrapolate=TRUE)
```

Arguments

x	Object of class "fv" or "rhohat".
...	Ignored.
value	Optional. Character string or character vector selecting one or more of the columns of x for use as the function value. See Details.
extrapolate	Logical, indicating whether to extrapolate the function outside the domain of x. See Details.

Details

A function value table (object of class "fv") is a convenient way of storing and plotting several different estimates of the same function. Objects of this class are returned by many commands in **spatstat**, such as Kest which returns an estimate of Ripley's K-function for a point pattern dataset.

Sometimes it is useful to convert the function value table to a function in the R language. This is done by as.function.fv. It converts an object x of class "fv" to an R function f.

If f <- as.function(x) then f is an R function that accepts a numeric argument and returns a corresponding value for the summary function by linear interpolation between the values in the table x.

Argument values lying outside the range of the table yield an NA value (if extrapolate=FALSE) or the function value at the nearest endpoint of the range (if extrapolate = TRUE). To apply different rules to the left and right extremes, use extrapolate=c(TRUE,FALSE) and so on.

Typically the table x contains several columns of function values corresponding to different edge corrections. Auxiliary information for the table identifies one of these columns as the *recommended value*. By default, the values of the function f <- as.function(x) are taken from this column of recommended values. This default can be changed using the argument value, which can be a character string or character vector of names of columns of x. Alternatively value can be one of the abbreviations used by fvnames.

If value specifies a single column of the table, then the result is a function f(r) with a single numeric argument r (with the same name as the orginal argument of the function table).

If value specifies several columns of the table, then the result is a function f(r,what) where r is the numeric argument and what is a character string identifying the column of values to be used.

The formal arguments of the resulting function are `f(r, what=value)`, which means that in a call to this function `f`, the permissible values of `what` are the entries of the original vector `value`; the default value of `what` is the first entry of `value`.

The command `as.function.fv` is a method for the generic command `as.function`.

Value

A `function(r)` or `function(r,what)` where `r` is the name of the original argument of the function table.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`fv`, `fv.object`, `fvnames`, `plot.fv`, `Kest`

Examples

```
K <- Kest(cells)
f <- as.function(K)
f
f(0.1)
g <- as.function(K, value=c("iso", "trans"))
g
g(0.1, "trans")
```

`as.function.im` *Convert Pixel Image to Function of Coordinates*

Description

Converts a pixel image to a function of the x and y coordinates.

Usage

```
## S3 method for class 'im'
as.function(x, ...)
```

Arguments

`x`	Pixel image (object of class `"im"`).
`...`	Ignored.

Details

This command converts a pixel image (object of class `"im"`) to a `function(x,y)` where the arguments `x` and `y` are (vectors of) spatial coordinates. This function returns the pixel values at the specified locations.

Value

A function in the R language, also belonging to the class "funxy".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

[.im

Examples

```
d <- density(cells)
f <- as.function(d)
f(0.1, 0.3)
```

---

as.function.leverage.ppm *Convert Leverage Object to Function of Coordinates*

Description

Converts an object of class "leverage.ppm" to a function of the x and y coordinates.

Usage

```
## S3 method for class 'leverage.ppm'
as.function(x, ...)
```

Arguments

x	Object of class "leverage.ppm" produced by leverage.ppm.
...	Ignored.

Details

An object of class "leverage.ppm" represents the leverage function of a fitted point process model. This command converts the object to a function(x,y) where the arguments x and y are (vectors of) spatial coordinates. This function returns the leverage values at the specified locations (calculated by referring to the nearest location where the leverage has been computed).

Value

A function in the R language, also belonging to the class "funxy".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

as.im.leverage.ppm

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X ~x+y)
lev <- leverage(fit)
f <- as.function(lev)

f(0.2, 0.3)  # evaluate at (x,y) coordinates
y <- f(X)    # evaluate at a point pattern
```

---

as.function.owin — *Convert Window to Indicator Function*

Description

Converts a spatial window to a function of the x and y coordinates returning the value 1 inside the window and 0 outside.

Usage

```
## S3 method for class 'owin'
as.function(x, ...)
```

Arguments

x	Pixel image (object of class "owin").
...	Ignored.

Details

This command converts a spatial window (object of class "owin") to a function(x,y) where the arguments x and y are (vectors of) spatial coordinates. This is the indicator function of the window: it returns the value 1 for locations inside the window, and returns 0 for values outside the window.

Value

A function in the R language.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

as.im.owin

Examples

```
W <- Window(humberside)
f <- as.function(W)
f(5000, 4500)
f(123456, 78910)
X <- runifpoint(5, Frame(humberside))
f(X)
```

`as.function.tess` *Convert a Tessellation to a Function*

Description

Convert a tessellation into a function of the x and y coordinates. The default function values are factor levels specifying which tile of the tessellation contains the point (x, y).

Usage

```
  ## S3 method for class 'tess'
as.function(x,...,values=NULL)
```

Arguments

`x`	A tessellation (object of class `"tess"`).
`values`	Optional. A vector giving the values of the function for each tile of `x`.
`...`	Ignored.

Details

This command converts a tessellation (object of class `"tess"`) to a `function(x,y)` where the arguments `x` and `y` are (vectors of) spatial coordinates. The corresponding function values are factor levels identifying which tile of the tessellation contains each point. Values are `NA` if the corresponding point lies outside the tessellation.

If the argument `values` is given, then it determines the value of the function in each tile of `x`.

Value

A function in the R language, also belonging to the class `"funxy"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`tileindex` for the low-level calculation of tile index.

`cut.ppp` and `split.ppp` to divide up the points of a point pattern according to a tessellation.

Examples

```
X <- runifpoint(7)
V <- dirichlet(X)
f <- as.function(V)
f(0.1, 0.4)
plot(f)
```

---

as.fv *Convert Data To Class fv*

Description

Converts data into a function table (an object of class "fv").

Usage

```
as.fv(x)

  ## S3 method for class 'fv'
as.fv(x)

  ## S3 method for class 'data.frame'
as.fv(x)

  ## S3 method for class 'matrix'
as.fv(x)

  ## S3 method for class 'fasp'
as.fv(x)

  ## S3 method for class 'minconfit'
as.fv(x)

  ## S3 method for class 'dppm'
as.fv(x)

  ## S3 method for class 'kppm'
as.fv(x)

  ## S3 method for class 'bw.optim'
as.fv(x)
```

Arguments

x	Data which will be converted into a function table

Details

This command converts data x, that could be interpreted as the values of a function, into a function value table (object of the class "fv" as described in fv.object). This object can then be plotted easily using plot.fv.

The dataset x may be any of the following:

- an object of class "fv";
- a matrix or data frame with at least two columns;
- an object of class "fasp", representing an array of "fv" objects.
- an object of class "minconfit", giving the results of a minimum contrast fit by the command mincontrast. The
- an object of class "kppm", representing a fitted Cox or cluster point process model, obtained from the model-fitting command kppm;
- an object of class "dppm", representing a fitted determinantal point process model, obtained from the model-fitting command dppm;
- an object of class "bw.optim", representing an optimal choice of smoothing bandwidth by a cross-validation method, obtained from commands like bw.diggle.

The function as.fv is generic, with methods for each of the classes listed above. The behaviour is as follows:

- If x is an object of class "fv", it is returned unchanged.
- If x is a matrix or data frame, the first column is interpreted as the function argument, and subsequent columns are interpreted as values of the function computed by different methods.
- If x is an object of class "fasp" representing an array of "fv" objects, these are combined into a single "fv" object.
- If x is an object of class "minconfit", or an object of class "kppm" or "dppm", the result is a function table containing the observed summary function and the best fit summary function.
- If x is an object of class "bw.optim", the result is a function table of the optimisation criterion as a function of the smoothing bandwidth.

Value

An object of class "fv" (see fv.object).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

Examples

```
r <- seq(0, 1, length=101)
x <- data.frame(r=r, y=r^2)
as.fv(x)
```

---

as.hyperframe *Convert Data to Hyperframe*

Description

Converts data from any suitable format into a hyperframe.

Usage

```
as.hyperframe(x, ...)

## Default S3 method:
as.hyperframe(x, ...)

## S3 method for class 'data.frame'
as.hyperframe(x, ..., stringsAsFactors=FALSE)

## S3 method for class 'hyperframe'
as.hyperframe(x, ...)

## S3 method for class 'listof'
as.hyperframe(x, ...)

## S3 method for class 'anylist'
as.hyperframe(x, ...)
```

Arguments

- `x` — Data in some other format.
- `...` — Optional arguments passed to `hyperframe`.
- `stringsAsFactors` — Logical. If `TRUE`, any column of the data frame `x` that contains character strings will be converted to a `factor`. If `FALSE`, no such conversion will occur.

Details

A hyperframe is like a data frame, except that its entries can be objects of any kind.

The generic function `as.hyperframe` converts any suitable kind of data into a hyperframe.

There are methods for the classes `data.frame`, `listof`, `anylist` and a default method, all of which convert data that is like a hyperframe into a hyperframe object. (The method for the class `listof` and `anylist` converts a list of objects, of arbitrary type, into a hyperframe with one column.) These methods do not discard any information.

There are also methods for other classes (see `as.hyperframe.ppx`) which extract the coordinates from a spatial dataset. These methods do discard some information.

Value

An object of class `"hyperframe"` created by `hyperframe`.

Conversion of Strings to Factors

Note that `as.hyperframe.default` will convert a character vector to a factor. It behaves like `as.data.frame`.

However `as.hyperframe.data.frame` does not convert strings to factors; it respects the structure of the data frame `x`.

The behaviour can be changed using the argument `stringsAsFactors`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

hyperframe, as.hyperframe.ppx

Examples

```
df <- data.frame(x=runif(4),y=letters[1:4])
as.hyperframe(df)

sims <- list()
for(i in 1:3) sims[[i]] <- rpoispp(42)
as.hyperframe(as.listof(sims))
as.hyperframe(as.solist(sims))
```

---

as.hyperframe.ppx *Extract coordinates and marks of multidimensional point pattern*

Description

Given any kind of spatial or space-time point pattern, extract the coordinates and marks of the points.

Usage

```
## S3 method for class 'ppx'
as.hyperframe(x, ...)
## S3 method for class 'ppx'
as.data.frame(x, ...)
## S3 method for class 'ppx'
as.matrix(x, ...)
```

Arguments

x	A general multidimensional space-time point pattern (object of class "ppx").
...	Ignored.

Details

An object of class "ppx" (see ppx) represents a marked point pattern in multidimensional space and/or time. There may be any number of spatial coordinates, any number of temporal coordinates, and any number of mark variables. The individual marks may be atomic (numeric values, factor values, etc) or objects of any kind.

The function as.hyperframe.ppx extracts the coordinates and the marks as a "hyperframe" (see hyperframe) with one row of data for each point in the pattern. This is a method for the generic function as.hyperframe.

The function as.data.frame.ppx discards those mark variables which are not atomic values, and extracts the coordinates and the remaining marks as a data.frame with one row of data for each point in the pattern. This is a method for the generic function as.data.frame.

Finally as.matrix(x) is equivalent to as.matrix(as.data.frame(x)) for an object of class "ppx". Be warned that, if there are any columns of non-numeric data (i.e. if there are mark variables that are factors), the result will be a matrix of character values.

Value

A hyperframe, data.frame or matrix as appropriate.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppx, hyperframe, as.hyperframe.

Examples

```
df <- data.frame(x=runif(4),y=runif(4),t=runif(4))
X <- ppx(data=df, coord.type=c("s","s","t"))
as.data.frame(X)
val <- runif(4)
E <- lapply(val, function(s) { rpoispp(s) })
hf <- hyperframe(t=val, e=as.listof(E))
Z <- ppx(data=hf, domain=c(0,1))
as.hyperframe(Z)
as.data.frame(Z)
```

---

as.im *Convert to Pixel Image*

Description

Converts various kinds of data to a pixel image

Usage

```
  as.im(X, ...)

  ## S3 method for class 'im'
as.im(X, W=NULL, ...,
        eps=NULL, dimyx=NULL, xy=NULL,
        na.replace=NULL)

  ## S3 method for class 'owin'
as.im(X, W=NULL, ...,
        eps=NULL, dimyx=NULL, xy=NULL,
        na.replace=NULL, value=1)
```

```
  ## S3 method for class 'matrix'
as.im(X, W=NULL, ...)

  ## S3 method for class 'tess'
as.im(X, W=NULL, ...,
        eps=NULL, dimyx=NULL, xy=NULL,
        na.replace=NULL)

  ## S3 method for class 'function'
as.im(X, W=NULL, ...,
        eps=NULL, dimyx=NULL, xy=NULL,
        na.replace=NULL, strict=FALSE)

  ## S3 method for class 'funxy'
as.im(X, W=Window(X), ...)

  ## S3 method for class 'distfun'
as.im(X, W=NULL, ...,
        eps=NULL, dimyx=NULL, xy=NULL,
        na.replace=NULL, approx=TRUE)

  ## S3 method for class 'nnfun'
as.im(X, W=NULL, ...,
        eps=NULL, dimyx=NULL, xy=NULL,
        na.replace=NULL)

  ## S3 method for class 'Smoothfun'
as.im(X, W=NULL, ...)

  ## S3 method for class 'leverage.ppm'
as.im(X, ..., what=c("smooth", "nearest"))

  ## S3 method for class 'data.frame'
as.im(X, ..., step, fatal=TRUE, drop=TRUE)

  ## Default S3 method:
as.im(X, W=NULL, ...,
        eps=NULL, dimyx=NULL, xy=NULL,
        na.replace=NULL)
```

Arguments

X	Data to be converted to a pixel image.
W	Window object which determines the spatial domain and pixel array geometry.
...	Additional arguments passed to X when X is a function.
eps,dimyx,xy	Optional parameters passed to as.mask which determine the pixel array geometry. See as.mask.
na.replace	Optional value to replace NA entries in the output image.
value	Optional. The value to be assigned to pixels inside the window, if X is a window.

`strict` Logical value indicating whether to match formal arguments of `X` when `X` is a function. If `strict=FALSE` (the default), all the `...` arguments are passed to `X`. If `strict=TRUE`, only named arguments are passed, and only if they match the names of formal arguments of `X`.

`step` Optional. A single number, or numeric vector of length 2, giving the grid step lengths in the x and y directions.

`fatal` Logical value indicating what to do if the resulting image would be too large for available memory. If `fatal=TRUE` (the default), an error occurs. If `fatal=FALSE`, a warning is issued and `NULL` is returned.

`drop` Logical value indicating what to do when `X` is a data frame with 3 columns. If `drop=TRUE` (the default), the result is a pixel image. If `drop=FALSE`, the result is a list containing one image.

`approx` Logical value indicating whether to compute an approximate result at faster speed, by using `distmap`, when `X` is a distance function.

`what` Character string (partially matched) specifying which image data should be extracted. See `plot.leverage.ppm` for explanation.

Details

This function converts the data `X` into a pixel image object of class `"im"` (see `im.object`). The function `as.im` is generic, with methods for the classes listed above.

Currently `X` may be any of the following:

- a pixel image object, of class `"im"`.
- a window object, of class `"owin"` (see `owin.object`). The result is an image with all pixel entries equal to `value` inside the window `X`, and `NA` outside.
- a matrix.
- a tessellation (object of class `"tess"`). The result is a factor-valued image, with one factor level corresponding to each tile of the tessellation. Pixels are classified according to the tile of the tessellation into which they fall.
- a single number (or a single logical, complex, factor or character value). The result is an image with all pixel entries equal to this constant value inside the window `W` (and `NA` outside, unless the argument `na.replace` is given). Argument `W` is required.
- a function of the form `function(x, y, ...)` which is to be evaluated to yield the image pixel values. In this case, the additional argument `W` must be present. This window will be converted to a binary image mask. Then the function `X` will be evaluated in the form `X(x, y, ...)` where `x` and `y` are **vectors** containing the x and y coordinates of all the pixels in the image mask, and `...` are any extra arguments given. This function must return a vector or factor of the same length as the input vectors, giving the pixel values.
- an object of class `"funxy"` representing a `function(x,y,...)`
- an object of class `"distfun"` representing a distance function (created by the command `distfun`).
- an object of class `"nnfun"` representing a nearest neighbour function (created by the command `nnfun`).
- a list with entries `x, y, z` in the format expected by the standard `R` functions `image.default` and `contour.default`. That is, `z` is a matrix of pixel values, `x` and `y` are vectors of x and y coordinates respectively, and `z[i,j]` is the pixel value for the location `(x[i],y[j])`.
- a point pattern (object of class `"ppp"`). See the separate documentation for `as.im.ppp`.

- A data frame with at least three columns. Columns named x, y and z, if present, will be assumed to contain the spatial coordinates and the pixel values, respectively. Otherwise the x and y coordinates will be taken from the first two columns of the data frame, and any remaining columns will be interpreted as pixel values.

The spatial domain (enclosing rectangle) of the pixel image is determined by the argument W. If W is absent, the spatial domain is determined by X. When X is a function, a matrix, or a single numerical value, W is required.

The pixel array dimensions of the final resulting image are determined by (in priority order)

- the argument eps, dimyx or xy if present;
- the pixel dimensions of the window W, if it is present and if it is a binary mask;
- the pixel dimensions of X if it is an image, a binary mask, or a list(x,y,z);
- the default pixel dimensions, controlled by spatstat.options.

Note that if eps, dimyx or xy is given, this will override the pixel dimensions of X if it has them. Thus, as.im can be used to change an image's pixel dimensions.

If the argument na.replace is given, then all NA entries in the image will be replaced by this value. The resulting image is then defined everwhere on the full rectangular domain, instead of a smaller window. Here na.replace should be a single value, of the same type as the other entries in the image.

If X is a pixel image that was created by an older version of **spatstat**, the command X <- as.im(X) will repair the internal format of X so that it conforms to the current version of **spatstat**.

If X is a data frame with m columns, then m-2 columns of data are interpreted as pixel values, yielding m-2 pixel images. The result of as.im.data.frame is a list of pixel images, belonging to the class "imlist". If m = 3 and drop=TRUE (the default), then the result is a pixel image rather than a list containing this image.

Value

A pixel image (object of class "im"), or a list of pixel images, or NULL if the conversion failed.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

Separate documentation for as.im.ppp

Examples

```
data(demopat)
# window object
W <- Window(demopat)
plot(W)
Z <- as.im(W)
image(Z)
# function
Z <- as.im(function(x,y) {x^2 + y^2}, unit.square())
image(Z)
# function with extra arguments
```

```
f <- function(x, y, x0, y0) {
    sqrt((x - x0)^2 + (y-y0)^2)
}
Z <- as.im(f, unit.square(), x0=0.5, y0=0.5)
image(Z)
# Revisit the Sixties
data(letterR)
Z <- as.im(f, letterR, x0=2.5, y0=2)
image(Z)
# usual convention in S
stuff <- list(x=1:10, y=1:10, z=matrix(1:100, nrow=10))
Z <- as.im(stuff)
# convert to finer grid
Z <- as.im(Z, dimyx=256)

# pixellate the Dirichlet tessellation
Di <- dirichlet(runifpoint(10))
plot(as.im(Di))
plot(Di, add=TRUE)

# as.im.data.frame is the reverse of as.data.frame.im
grad <- bei.extra$grad
slopedata <- as.data.frame(grad)
slope <- as.im(slopedata)
unitname(slope) <- c("metre","metres")
all.equal(slope, grad) # TRUE
```

---

as.interact *Extract Interaction Structure*

Description

Extracts the interpoint interaction structure from a point pattern model.

Usage

```
as.interact(object)
## S3 method for class 'fii'
as.interact(object)
## S3 method for class 'interact'
as.interact(object)
## S3 method for class 'ppm'
as.interact(object)
```

Arguments

object	A fitted point process model (object of class "ppm") or an interpoint interaction structure (object of class "interact").

Details

The function `as.interact` extracts the interpoint interaction structure from a suitable object.

An object of class "interact" describes an interpoint interaction structure, before it has been fitted to point pattern data. The irregular parameters of the interaction (such as the interaction range) are fixed, but the regular parameters (such as interaction strength) are undetermined. Objects of this class are created by the functions `Poisson`, `Strauss` and so on. The main use of such objects is in a call to `ppm`.

The function `as.interact` is generic, with methods for the classes `"ppm"`, `"fii"` and `"interact"`. The result is an object of class `"interact"` which can be printed.

Value

An object of class `"interact"` representing the interpoint interaction. This object can be printed and plotted.

Note on parameters

This function does **not** extract the fitted coefficients of the interaction. To extract the fitted interaction including the fitted coefficients, use `fitin`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`fitin`, `ppm`.

Examples

```
data(cells)
model <- ppm(cells, ~1, Strauss(0.07))
f <- as.interact(model)
f
```

`as.layered` *Convert Data To Layered Object*

Description

Converts spatial data into a layered object.

Usage

```
 as.layered(X)

 ## Default S3 method:
as.layered(X)

 ## S3 method for class 'ppp'
as.layered(X)

 ## S3 method for class 'splitppp'
as.layered(X)
```

```
## S3 method for class 'solist'
as.layered(X)

## S3 method for class 'listof'
as.layered(X)

## S3 method for class 'msr'
as.layered(X)
```

Arguments

X	Some kind of spatial data.

Details

This function converts the object X into an object of class "layered".

The argument X should contain some kind of spatial data such as a point pattern, window, or pixel image.

If X is a simple object then it will be converted into a layered object containing only one layer which is equivalent to X.

If X can be interpreted as consisting of multiple layers of data, then the result will be a layered object consisting of these separate layers of data.

- if X is a list of class "listof" or "solist", then as.layered(X) consists of several layers, one for each entry in the list X;
- if X is a multitype point pattern, then as.layered(X) consists of several layers, each containing the sub-pattern consisting of points of one type;
- if X is a vector-valued measure, then as.layered(X) consists of several layers, each containing a scalar-valued measure.

Value

An object of class "layered" (see layered).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

layered, split.ppp

Examples

```
as.layered(cells)
as.layered(amacrine)

P <- rpoispp(100)
fit <- ppm(P ~ x+y)
rs <- residuals(fit, type="score")
as.layered(rs)
```

`as.linfun` *Convert Data to a Function on a Linear Network*

Description

Convert some kind of data to an object of class "linfun" representing a function on a linear network.

Usage

```
  as.linfun(X, ...)

  ## S3 method for class 'linim'
as.linfun(X, ...)

  ## S3 method for class 'lintess'
as.linfun(X, ..., values, navalue=NA)
```

Arguments

X	Some kind of data to be converted.
...	Other arguments passed to methods.
values	Optional. Vector of function values, one entry associated with each tile of the tessellation.
navalue	Optional. Function value associated with locations that do not belong to a tile of the tessellation.

Details

An object of class "linfun" represents a function defined on a linear network.

The function `as.linfun` is generic. The method `as.linfun.linim` converts objects of class "linim" (pixel images on a linear network) to functions on the network.

The method `as.linfun.lintess` converts a tessellation on a linear network into a function with a different value on each tile of the tessellation. If the argument `values` is missing or null, then the function returns factor values identifying which tile contains each given point. If `values` is given, it should be a vector with one entry for each tile of the tessellation: any point lying in tile number `i` will return the value `v[i]`.

Value

Object of class "linfun".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

linfun

Examples

```
X <- runiflpp(2, simplenet)
Y <- runiflpp(5, simplenet)

# image on network
D <- density(Y, 0.1, verbose=FALSE)

f <- as.linfun(D)
f
f(X)

# tessellation on network
Z <- lineardirichlet(Y)
g <- as.linfun(Z)
g(X)
h <- as.linfun(Z, values = runif(5))
h(X)
```

`as.linim` *Convert to Pixel Image on Linear Network*

Description

Converts various kinds of data to a pixel image on a linear network.

Usage

```
  as.linim(X, ...)

  ## S3 method for class 'linim'
as.linim(X, ...)

  ## Default S3 method:
as.linim(X, L, ...,
                              eps = NULL, dimyx = NULL, xy = NULL,
                              delta=NULL)

  ## S3 method for class 'linfun'
as.linim(X, L=domain(X), ...,
                              eps = NULL, dimyx = NULL, xy = NULL,
                              delta=NULL)
```

Arguments

`X`	Data to be converted to a pixel image on a linear network.
`L`	Linear network (object of class `"linnet"`).
`...`	Additional arguments passed to `X` when `X` is a function.
`eps,dimyx,xy`	Optional arguments passed to `as.mask` to control the pixel resolution.
`delta`	Optional. Numeric value giving the approximate distance (in coordinate units) between successive sample points along each segment of the network.

Details

This function converts the data X into a pixel image on a linear network, an object of class "linim" (see linim).

The argument X may be any of the following:

- a function on a linear network, an object of class "linfun".
- a pixel image on a linear network, an object of class "linim".
- a pixel image, an object of class "im".
- any type of data acceptable to as.im, such as a function, numeric value, or window.

First X is converted to a pixel image object Y (object of class "im"). The conversion is performed by as.im. The arguments eps, dimyx and xy determine the pixel resolution.

Next Y is converted to a pixel image on a linear network using linim. The argument L determines the linear network. If L is missing or NULL, then X should be an object of class "linim", and L defaults to the linear network on which X is defined.

In addition to converting the function to a pixel image, the algorithm also generates a fine grid of sample points evenly spaced along each segment of the network (with spacing at most delta coordinate units). The function values at these sample points are stored in the resulting object as a data frame (the argument df of linim). This mechanism allows greater accuracy for some calculations (such as integral.linim).

Value

An image object on a linear network; an object of class "linim".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

as.im

Examples

```
f <- function(x,y){ x + y }
plot(as.linim(f, simplenet))
```

---

as.linnet.linim *Extract Linear Network from Data on a Linear Network*

Description

Given some kind of data on a linear network, the command as.linnet extracts the linear network itself.

Usage

```
## S3 method for class 'linim'
as.linnet(X, ...)

## S3 method for class 'linfun'
as.linnet(X, ...)

## S3 method for class 'lintess'
as.linnet(X, ...)

## S3 method for class 'lpp'
as.linnet(X, ..., fatal=TRUE, sparse)
```

Arguments

X	Data on a linear network. A point pattern (class "lpp"), pixel image (class "linim"), function (class "linfun") or tessellation (class "lintess") on a linear network.
...	Ignored.
fatal	Logical value indicating whether data in the wrong format should lead to an error (fatal=TRUE) or a warning (fatal=FALSE).
sparse	Logical value indicating whether to use a sparse matrix representation, as explained in linnet. Default is to keep the same representation as in X.

Details

These are methods for the generic as.linnet for various classes.

The network on which the data are defined is extracted.

Value

A linear network (object of class "linnet").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

linnet, methods.linnet.

Examples

```
# make some data
xcoord <- linfun(function(x,y,seg,tp) { x }, simplenet)
as.linnet(xcoord)
X <- as.linim(xcoord)
as.linnet(X)
```

as.linnet.psp *Convert Line Segment Pattern to Linear Network*

Description

Converts a line segment pattern to a linear network.

Usage

```
## S3 method for class 'psp'
as.linnet(X, ..., eps, sparse=FALSE)
```

Arguments

X	Line segment pattern (object of class "psp").
...	Ignored.
eps	Optional. Distance threshold. If two segment endpoints are closer than eps units apart, they will be treated as the same point, and will become a single vertex in the linear network.
sparse	Logical value indicating whether to use a sparse matrix representation, as explained in linnet.

Details

This command converts any collection of line segments into a linear network by guessing the connectivity of the network, using the distance threshold eps.

If any segments in X cross over each other, they are first cut into pieces using selfcut.psp.

Then any pair of segment endpoints lying closer than eps units apart, is treated as a single vertex. The linear network is then constructed using linnet.

It would be wise to check the result by plotting the degree of each vertex, as shown in the Examples.

If X has marks, then these are stored in the resulting linear network Y <- as.linnet(X), and can be extracted as marks(as.psp(Y)) or marks(Y$lines).

Value

A linear network (object of class "linnet").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

linnet, selfcut.psp, methods.linnet.

Examples

```
# make some data
A <- psp(0.09, 0.55, 0.79, 0.80, window=owin())
B <- superimpose(A, as.psp(simplenet))

# convert to a linear network
D <- as.linnet(B)

# check validity
D
plot(D)
text(vertices(D), labels=vertexdegree(D))
```

---

as.lpp *Convert Data to a Point Pattern on a Linear Network*

Description

Convert various kinds of data to a point pattern on a linear network.

Usage

```
as.lpp(x=NULL, y=NULL, seg=NULL, tp=NULL, ...,
       marks=NULL, L=NULL, check=FALSE, sparse)
```

Arguments

`x,y`	Vectors of cartesian coordinates, or any data acceptable to `xy.coords`. Alternatively `x` can be a point pattern on a linear network (object of class `"lpp"`) or a planar point pattern (object of class `"ppp"`).
`seg,tp`	Optional local coordinates. Vectors of the same length as `x,y`. See Details.
`...`	Ignored.
`marks`	Optional marks for the point pattern. A vector or factor with one entry for each point, or a data frame or hyperframe with one row for each point.
`L`	Linear network (object of class `"linnet"`) on which the points lie.
`check`	Logical. Whether to check the validity of the spatial coordinates.
`sparse`	Optional logical value indicating whether to store the linear network data in a sparse matrix representation or not. See `linnet`.

Details

This function converts data in various formats into a point pattern on a linear network (object of class `"lpp"`).

The possible formats are:

- `x` is already a point pattern on a linear network (object of class `"lpp"`). Then `x` is returned unchanged.
- `x` is a planar point pattern (object of class `"ppp"`). Then `x` is converted to a point pattern on the linear network `L` using `lpp`.

- `x,y,seg,tp` are vectors of equal length. These specify that the `i`th point has Cartesian coordinates `(x[i],y[i])`, and lies on segment number `seg[i]` of the network `L`, at a fractional position `tp[i]` along that segment (with `tp=0` representing one endpoint and `tp=1` the other endpoint of the segment).
- `x,y` are missing and `seg,tp` are vectors of equal length as described above.
- `seg,tp` are `NULL`, and `x,y` are data in a format acceptable to `xy.coords` specifying the Cartesian coordinates.

Value

A point pattern on a linear network (object of class `"lpp"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`lpp`.

Examples

```
A <- as.psp(simplenet)
X <- runifpointOnLines(10, A)
is.ppp(X)
Y <- as.lpp(X, L=simplenet)
```

`as.mask` *Pixel Image Approximation of a Window*

Description

Obtain a discrete (pixel image) approximation of a given window

Usage

```
as.mask(w, eps=NULL, dimyx=NULL, xy=NULL)
```

Arguments

`w`	A window (object of class `"owin"`) or data acceptable to `as.owin`.
`eps`	(optional) width and height of pixels.
`dimyx`	(optional) pixel array dimensions
`xy`	(optional) data containing pixel coordinates

Details

This function generates a rectangular grid of locations in the plane, tests whether each of these locations lies inside the window `w`, and stores the results as a binary pixel image or 'mask' (an object of class `"owin"`, see `owin.object`).

The most common use of this function is to approximate the shape of another window `w` by a binary pixel image. In this case, we will usually want to have a very fine grid of pixels.

This function can also be used to generate a coarsely-spaced grid of locations inside a window, for purposes such as subsampling and prediction.

The grid spacing and location are controlled by the arguments `eps`, `dimyx` and `xy`, which are mutually incompatible.

If `eps` is given, then it determines the grid spacing. If `eps` is a single number, then the grid spacing will be approximately `eps` in both the x and y directions. If `eps` is a vector of length 2, then the grid spacing will be approximately `eps[1]` in the x direction and `eps[2]` in the y direction.

If `dimyx` is given, then the pixel grid will be an $m \times n$ rectangular grid where m, n are given by `dimyx[2]`, `dimyx[1]` respectively. **Warning:** `dimyx[1]` is the number of pixels in the y direction, and `dimyx[2]` is the number in the x direction.

If `xy` is given, then this should be some kind of data specifing the coordinates of a pixel grid. It may be

- a list or structure containing elements `x` and `y` which are numeric vectors of equal length. These will be taken as x and `y` coordinates of the margins of the grid. The pixel coordinates will be generated from these two vectors.
- a pixel image (object of class `"im"`).
- a window (object of class `"owin"`) which is of type `"mask"` so that it contains pixel coordinates.

If `xy` is given, `w` may be omitted.

If neither `eps` nor `dimyx` nor `xy` is given, the pixel raster dimensions are obtained from `spatstat.options("npixel")`.

There is no inverse of this function. However, the function `as.polygonal` will compute a polygonal approximation of a binary mask.

Value

A window (object of class `"owin"`) of type `"mask"` representing a binary pixel image.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`owin.object`, `as.rectangle`, `as.polygonal`, `spatstat.options`

Examples

```
w <- owin(c(0,10),c(0,10), poly=list(x=c(1,2,3,2,1), y=c(2,3,4,6,7)))
## Not run: plot(w)
m <- as.mask(w)
## Not run: plot(m)
```

```
x <- 1:9
y <- seq(0.25, 9.75, by=0.5)
m <- as.mask(w, xy=list(x=x, y=y))
```

---

as.mask.psp *Convert Line Segment Pattern to Binary Pixel Mask*

Description

Converts a line segment pattern to a binary pixel mask by determining which pixels intersect the lines.

Usage

```
as.mask.psp(x, W=NULL, ...)
```

Arguments

x	Line segment pattern (object of class "psp").
W	Optional window (object of class "owin") determining the pixel raster.
...	Optional extra arguments passed to as.mask to determine the pixel resolution.

Details

This function converts a line segment pattern to a binary pixel mask by determining which pixels intersect the lines.

The pixel raster is determined by W and the optional arguments .... If W is missing or NULL, it defaults to the window containing x. Then W is converted to a binary pixel mask using as.mask. The arguments ... are passed to as.mask to control the pixel resolution.

Value

A window (object of class "owin") which is a binary pixel mask (type "mask").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

pixellate.psp, as.mask.

Use pixellate.psp if you want to measure the length of line in each pixel.

Examples

```
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(as.mask.psp(X))
plot(X, add=TRUE, col="red")
```

`as.matrix.im` *Convert Pixel Image to Matrix or Array*

Description

Converts a pixel image to a matrix or an array.

Usage

```
  ## S3 method for class 'im'
as.matrix(x, ...)
  ## S3 method for class 'im'
as.array(x, ...)
```

Arguments

x	A pixel image (object of class "im").
...	See below.

Details

The function `as.matrix.im` converts the pixel image `x` into a matrix containing the pixel values. It is handy when you want to extract a summary of the pixel values. See the Examples.

The function `as.array.im` converts the pixel image to an array. By default this is a three-dimensional array of dimension n by m by 1. If the extra arguments `...` are given, they will be passed to `array`, and they may change the dimensions of the array.

Value

A matrix or array.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`as.matrix.owin`

Examples

```
# artificial image
Z <- setcov(square(1))

M <- as.matrix(Z)

median(M)

## Not run:
# plot the cumulative distribution function of pixel values
plot(ecdf(as.matrix(Z)))
```

```
## End(Not run)
```

as.matrix.owin *Convert Pixel Image to Matrix*

Description

Converts a pixel image to a matrix.

Usage

```
  ## S3 method for class 'owin'
as.matrix(x, ...)
```

Arguments

x	A window (object of class "owin").
...	Arguments passed to as.mask to control the pixel resolution.

Details

The function as.matrix.owin converts a window to a logical matrux.

It first converts the window x into a binary pixel mask using as.mask. It then extracts the pixel entries as a logical matrix.

The resulting matrix has entries that are TRUE if the corresponding pixel is inside the window, and FALSE if it is outside.

The function as.matrix is generic. The function as.matrix.owin is the method for windows (objects of class "owin").

Use as.im to convert a window to a pixel image.

Value

A logical matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

as.matrix.im, as.im

Examples

```
 m <- as.matrix(letterR)
```

as.owin *Convert Data To Class owin*

Description

Converts data specifying an observation window in any of several formats, into an object of class "owin".

Usage

```
 as.owin(W, ..., fatal=TRUE)

 ## S3 method for class 'owin'
as.owin(W, ..., fatal=TRUE)

 ## S3 method for class 'ppp'
as.owin(W, ..., fatal=TRUE)

 ## S3 method for class 'ppm'
as.owin(W, ..., from=c("points", "covariates"), fatal=TRUE)

 ## S3 method for class 'kppm'
as.owin(W, ..., from=c("points", "covariates"), fatal=TRUE)

 ## S3 method for class 'dppm'
as.owin(W, ..., from=c("points", "covariates"), fatal=TRUE)

 ## S3 method for class 'lpp'
as.owin(W, ..., fatal=TRUE)

 ## S3 method for class 'lppm'
as.owin(W, ..., fatal=TRUE)

 ## S3 method for class 'msr'
as.owin(W, ..., fatal=TRUE)

 ## S3 method for class 'psp'
as.owin(W, ..., fatal=TRUE)

 ## S3 method for class 'quad'
as.owin(W, ..., fatal=TRUE)

 ## S3 method for class 'quadratcount'
as.owin(W, ..., fatal=TRUE)

 ## S3 method for class 'quadrattest'
as.owin(W, ..., fatal=TRUE)

 ## S3 method for class 'tess'
as.owin(W, ..., fatal=TRUE)
```

```
## S3 method for class 'im'
as.owin(W, ..., fatal=TRUE)

## S3 method for class 'layered'
as.owin(W, ..., fatal=TRUE)

## S3 method for class 'data.frame'
as.owin(W, ..., step, fatal=TRUE)

## S3 method for class 'distfun'
as.owin(W, ..., fatal=TRUE)

## S3 method for class 'nnfun'
as.owin(W, ..., fatal=TRUE)

## S3 method for class 'funxy'
as.owin(W, ..., fatal=TRUE)

## S3 method for class 'boxx'
as.owin(W, ..., fatal=TRUE)

## S3 method for class 'rmhmodel'
as.owin(W, ..., fatal=FALSE)

## S3 method for class 'leverage.ppm'
as.owin(W, ..., fatal=TRUE)

## S3 method for class 'influence.ppm'
as.owin(W, ..., fatal=TRUE)

## Default S3 method:
as.owin(W, ..., fatal=TRUE)
```

Arguments

`W`	Data specifying an observation window, in any of several formats described under *Details* below.
`fatal`	Logical flag determining what to do if the data cannot be converted to an observation window. See Details.
`...`	Ignored.
`from`	Character string. See Details.
`step`	Optional. A single number, or numeric vector of length 2, giving the grid step lengths in the x and y directions.

Details

The class `"owin"` is a way of specifying the observation window for a point pattern. See `owin.object` for an overview.

This function converts data in any of several formats into an object of class `"owin"` for use by the **spatstat** package. The function `as.owin` is generic, with methods for different classes of objects, and a default method.

The argument `W` may be

- an object of class "owin"
- a structure with entries xrange, yrange specifying the x and y dimensions of a rectangle
- a four-element vector (interpreted as (xmin, xmax, ymin, ymax)) specifying the x and y dimensions of a rectangle
- a structure with entries xl, xu, yl, yu specifying the x and y dimensions of a rectangle as (xmin, xmax) = (xl, xu) and (ymin, ymax) = (yl, yu). This will accept objects of class spp used in the Venables and Ripley **spatial** library.
- an object of class "ppp" representing a point pattern. In this case, the object's window structure will be extracted.
- an object of class "psp" representing a line segment pattern. In this case, the object's window structure will be extracted.
- an object of class "tess" representing a tessellation. In this case, the object's window structure will be extracted.
- an object of class "quad" representing a quadrature scheme. In this case, the window of the data component will be extracted.
- an object of class "im" representing a pixel image. In this case, a window of type "mask" will be returned, with the same pixel raster coordinates as the image. An image pixel value of NA, signifying that the pixel lies outside the window, is transformed into the logical value FALSE, which is the corresponding convention for window masks.
- an object of class "ppm", "kppm" or "dppm" representing a fitted point process model. In this case, if from="data" (the default), as.owin extracts the original point pattern data to which the model was fitted, and returns the observation window of this point pattern. If from="covariates" then as.owin extracts the covariate images to which the model was fitted, and returns a binary mask window that specifies the pixel locations.
- an object of class "lpp" representing a point pattern on a linear network. In this case, as.owin extracts the linear network and returns a window containing this network.
- an object of class "lppm" representing a fitted point process model on a linear network. In this case, as.owin extracts the linear network and returns a window containing this network.
- A data.frame with exactly three columns. Each row of the data frame corresponds to one pixel. Each row contains the x and y coordinates of a pixel, and a logical value indicating whether the pixel lies inside the window.
- A data.frame with exactly two columns. Each row of the data frame contains the x and y coordinates of a pixel that lies inside the window.
- an object of class "distfun", "nnfun" or "funxy" representing a function of spatial location, defined on a spatial domain. The spatial domain of the function will be extracted.
- an object of class "rmhmodel" representing a point process model that can be simulated using rmh. The window (spatial domain) of the model will be extracted. The window may be NULL in some circumstances (indicating that the simulation window has not yet been determined). This is not treated as an error, because the argument fatal defaults to FALSE for this method.
- an object of class "layered" representing a list of spatial objects. See layered. In this case, as.owin will be applied to each of the objects in the list, and the union of these windows will be returned.

If the argument W is not in one of these formats and cannot be converted to a window, then an error will be generated (if fatal=TRUE) or a value of NULL will be returned (if fatal=FALSE).

When W is a data frame, the argument step can be used to specify the pixel grid spacing; otherwise, the spacing will be guessed from the data.

Value

An object of class "owin" (see owin.object) specifying an observation window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

owin.object, owin

Examples

```
w <- as.owin(c(0,1,0,1))
w <- as.owin(list(xrange=c(0,5),yrange=c(0,10)))
# point pattern
data(demopat)
w <- as.owin(demopat)
# image
Z <- as.im(function(x,y) { x + 3}, unit.square())
w <- as.owin(Z)

# Venables & Ripley 'spatial' package
require(spatial)
towns <- ppinit("towns.dat")
w <- as.owin(towns)
detach(package:spatial)
```

---

as.polygonal *Convert a Window to a Polygonal Window*

Description

Given a window W of any geometric type (rectangular, polygonal or binary mask), this function returns a polygonal window that represents the same spatial domain.

Usage

```
as.polygonal(W, repair=FALSE)
```

Arguments

W	A window (object of class "owin").
repair	Logical value indicating whether to check the validity of the polygon data and repair it, if W is already a polygonal window.

Details

Given a window W of any geometric type (rectangular, polygonal or binary mask), this function returns a polygonal window that represents the same spatial domain.

If W is a rectangle, it is converted to a polygon with 4 vertices.

If W is already polygonal, it is returned unchanged, by default. However if repair=TRUE then the validity of the polygonal coordinates will be checked (for example to check the boundary is not self-intersecting) and repaired if necessary, so that the result could be different from W.

If W is a binary mask, then each pixel in the mask is replaced by a small square or rectangle, and the union of these squares or rectangles is computed. The result is a polygonal window that has only horizontal and vertical edges. (Use simplify.owin to remove the staircase appearance, if desired).

Value

A polygonal window (object of class "owin" and of type "polygonal").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

owin, as.owin, as.mask, simplify.owin

Examples

```
data(letterR)
m <- as.mask(letterR, dimyx=32)
p <- as.polygonal(m)
if(interactive()) {
   plot(m)
   plot(p, add=TRUE, lwd=2)
}
```

---

as.ppm *Extract Fitted Point Process Model*

Description

Extracts the fitted point process model from some kind of fitted model.

Usage

```
as.ppm(object)

## S3 method for class 'ppm'
as.ppm(object)

## S3 method for class 'profilepl'
as.ppm(object)
```

```
## S3 method for class 'kppm'
as.ppm(object)

## S3 method for class 'dppm'
as.ppm(object)
```

Arguments

object — An object that includes a fitted Poisson or Gibbs point process model. An object of class "ppm", "profilepl", "kppm" or "dppm" or possibly other classes.

Details

The function as.ppm extracts the fitted point process model (of class "ppm") from a suitable object.

The function as.ppm is generic, with methods for the classes "ppm", "profilepl", "kppm" and "dppm", and possibly for other classes.

For the class "profilepl" of models fitted by maximum profile pseudolikelihood, the method as.ppm.profilepl extracts the fitted point process model (with the optimal values of the irregular parameters).

For the class "kppm" of models fitted by minimum contrast (or Palm or composite likelihood) using Waagepetersen's two-step estimation procedure (see kppm), the method as.ppm.kppm extracts the Poisson point process model that is fitted in the first stage of the procedure.

The behaviour for the class "dppm" is analogous to the "kppm" case above.

Value

An object of class "ppm".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

ppm, profilepl.

Examples

```
# fit a model by profile maximum pseudolikelihood
rvals <- data.frame(r=(1:10)/100)
pfit <- profilepl(rvals, Strauss, cells, ~1)
# extract the fitted model
fit <- as.ppm(pfit)
```

as.ppp *Convert Data To Class ppp*

Description

Tries to coerce any reasonable kind of data to a spatial point pattern (an object of class "ppp") for use by the **spatstat** package).

Usage

```
  as.ppp(X, ..., fatal=TRUE)

  ## S3 method for class 'ppp'
as.ppp(X, ..., fatal=TRUE)

  ## S3 method for class 'psp'
as.ppp(X, ..., fatal=TRUE)

  ## S3 method for class 'quad'
as.ppp(X, ..., fatal=TRUE)

  ## S3 method for class 'matrix'
as.ppp(X, W=NULL, ..., fatal=TRUE)

  ## S3 method for class 'data.frame'
as.ppp(X, W=NULL, ..., fatal=TRUE)

  ## S3 method for class 'influence.ppm'
as.ppp(X, ...)

  ## Default S3 method:
as.ppp(X, W=NULL, ..., fatal=TRUE)
```

Arguments

X	Data which will be converted into a point pattern
W	Data which define a window for the pattern, when X does not contain a window. (Ignored if X contains window information.)
...	Ignored.
fatal	Logical value specifying what to do if the data cannot be converted. See Details.

Details

Converts the dataset X to a point pattern (an object of class "ppp"; see ppp.object for an overview).

This function is normally used to convert an existing point pattern dataset, stored in another format, to the "ppp" format. To create a new point pattern from raw data such as x, y coordinates, it is normally easier to use the creator function ppp.

The function as.ppp is generic, with methods for the classes "ppp", "psp", "quad", "matrix", "data.frame" and a default method.

The dataset X may be:

- an object of class "ppp"
- an object of class "psp"
- a point pattern object created by the **spatial** library
- an object of class "quad" representing a quadrature scheme (see quad.object)
- a matrix or data frame with at least two columns
- a structure with entries x, y which are numeric vectors of equal length
- a numeric vector of length 2, interpreted as the coordinates of a single point.

In the last three cases, we need the second argument W which is converted to a window object by the function as.owin. In the first four cases, W will be ignored.

If X is a line segment pattern (an object of class psp) the point pattern returned consists of the endpoints of the segments. If X is marked then the point pattern returned will also be marked, the mark associated with a point being the mark of the segment of which that point was an endpoint.

If X is a matrix or data frame, the first and second columns will be interpreted as the x and y coordinates respectively. Any additional columns will be interpreted as marks.

The argument fatal indicates what to do when W is missing and X contains no information about the window. If fatal=TRUE, a fatal error will be generated; if fatal=FALSE, the value NULL is returned.

In the **spatial** library, a point pattern is represented in either of the following formats:

- (in **spatial** versions 1 to 6) a structure with entries x, y xl, xu, yl, yu
- (in **spatial** version 7) a structure with entries x, y and area, where area is a structure with entries xl, xu, yl, yu

where x and y are vectors of equal length giving the point coordinates, and xl, xu, yl, yu are numbers giving the dimensions of a rectangular window.

Point pattern datasets can also be created by the function ppp.

Value

An object of class "ppp" (see ppp.object) describing the point pattern and its window of observation. The value NULL may also be returned; see Details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

ppp, ppp.object, as.owin, owin.object

Examples

```
xy <- matrix(runif(40), ncol=2)
pp <- as.ppp(xy, c(0,1,0,1))

# Venables-Ripley format
# check for 'spatial' package
spatialpath <- system.file(package="spatial")
if(nchar(spatialpath) > 0) {
  require(spatial)
```

```
    towns <- ppinit("towns.dat")
    pp <- as.ppp(towns) # converted to our format
    detach(package:spatial)
  }

  xyzt <- matrix(runif(40), ncol=4)
  Z <- as.ppp(xyzt, square(1))
```

---

as.psp *Convert Data To Class psp*

Description

Tries to coerce any reasonable kind of data object to a line segment pattern (an object of class "psp") for use by the **spatstat** package.

Usage

```
  as.psp(x, ..., from=NULL, to=NULL)

  ## S3 method for class 'psp'
as.psp(x, ..., check=FALSE, fatal=TRUE)

  ## S3 method for class 'data.frame'
as.psp(x, ..., window=NULL, marks=NULL,
      check=spatstat.options("checksegments"), fatal=TRUE)

  ## S3 method for class 'matrix'
as.psp(x, ..., window=NULL, marks=NULL,
       check=spatstat.options("checksegments"), fatal=TRUE)

  ## Default S3 method:
as.psp(x, ..., window=NULL, marks=NULL,
       check=spatstat.options("checksegments"), fatal=TRUE)
```

Arguments

x	Data which will be converted into a line segment pattern
window	Data which define a window for the pattern.
...	Ignored.
marks	(Optional) vector or data frame of marks for the pattern
check	Logical value indicating whether to check the validity of the data, e.g. to check that the line segments lie inside the window.
fatal	Logical value. See Details.
from,to	Point patterns (object of class "ppp") containing the first and second endpoints (respectively) of each segment. Incompatible with x.

Details

Converts the dataset x to a line segment pattern (an object of class "psp"; see psp.object for an overview).

This function is normally used to convert an existing line segment pattern dataset, stored in another format, to the "psp" format. To create a new point pattern from raw data such as x, y coordinates, it is normally easier to use the creator function psp.

The dataset x may be:

- an object of class "psp"
- a data frame with at least 4 columns
- a structure (list) with elements named x0, y0, x1, y1 or elements named xmid, ymid, length, angle and possibly a fifth element named marks

If x is a data frame the interpretation of its columns is as follows:

- If there are columns named x0, y0, x1, y1 then these will be interpreted as the coordinates of the endpoints of the segments and used to form the ends component of the psp object to be returned.
- If there are columns named xmid, ymid, length, angle then these will be interpreted as the coordinates of the segment midpoints, the lengths of the segments, and the orientations of the segments in radians and used to form the ends component of the psp object to be returned.
- If there is a column named marks then this will be interpreted as the marks of the pattern provided that the argument marks of this function is NULL. If argument marks is not NULL then the value of this argument is taken to be the marks of the pattern and the column named marks is ignored (with a warning). In either case the column named marks is deleted and omitted from further consideration.
- If there is no column named marks and if the marks argument of this function is NULL, and if after interpreting 4 columns of x as determining the ends component of the psp object to be returned, there remain other columns of x, then these remaining columns will be taken to form a data frame of marks for the psp object to be returned.

If x is a structure (list) with elements named x0, y0, x1, y1, marks or xmid, ymid, length, angle, marks, then the element named marks will be interpreted as the marks of the pattern provide that the argument marks of this function is NULL. If this argument is non-NULL then it is interpreted as the marks of the pattern and the element marks of x is ignored — with a warning.

Alternatively, you may specify two point patterns from and to containing the first and second endpoints of the line segments.

The argument window is converted to a window object by the function as.owin.

The argument fatal indicates what to do when the data cannot be converted to a line segment pattern. If fatal=TRUE, a fatal error will be generated; if fatal=FALSE, the value NULL is returned.

The function as.psp is generic, with methods for the classes "psp", "data.frame", "matrix" and a default method.

Point pattern datasets can also be created by the function psp.

Value

An object of class "psp" (see psp.object) describing the line segment pattern and its window of observation. The value NULL may also be returned; see Details.

Warnings

If only a proper subset of the names x0,y0,x1,y1 or xmid,ymid,length,angle appear amongst the names of the columns of x where x is a data frame, then these special names are ignored.

For example if the names of the columns were xmid,ymid,length,degrees, then these columns would be interpreted as if the represented x0,y0,x1,y1 in that order.

Whether it gets used or not, column named marks is *always* removed from x before any attempt to form the ends component of the psp object that is returned.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

psp, psp.object, as.owin, owin.object.

See edges for extracting the edges of a polygonal window as a "psp" object.

Examples

```
mat <- matrix(runif(40), ncol=4)
mx <- data.frame(v1=sample(1:4,10,TRUE),
                 v2=factor(sample(letters[1:4],10,TRUE),levels=letters[1:4]))
a <- as.psp(mat, window=owin(),marks=mx)
mat <- cbind(as.data.frame(mat),mx)
b <- as.psp(mat, window=owin()) # a and b are identical.
stuff <- list(xmid=runif(10),
              ymid=runif(10),
              length=rep(0.1, 10),
              angle=runif(10, 0, 2 * pi))
a <- as.psp(stuff, window=owin())
b <- as.psp(from=runifpoint(10), to=runifpoint(10))
```

---

as.rectangle *Window Frame*

Description

Extract the window frame of a window or other spatial dataset

Usage

```
as.rectangle(w, ...)
```

Arguments

w	A window, or a dataset that has a window. Either a window (object of class "owin"), a pixel image (object of class "im") or other data determining such a window.
...	Optional. Auxiliary data to help determine the window. If w does not belong to a recognised class, the arguments w and ... are passed to as.owin to determine the window.

Details

This function is the quickest way to determine a bounding rectangle for a spatial dataset.

If `w` is a window, the function just extracts the outer bounding rectangle of `w` as given by its elements `xrange,yrange`.

The function can also be applied to any spatial dataset that has a window: for example, a point pattern (object of class `"ppp"`) or a line segment pattern (object of class `"psp"`). The bounding rectangle of the window of the dataset is extracted.

Use the function `boundingbox` to compute the *smallest* bounding rectangle of a dataset.

Value

A window (object of class `"owin"`) of type `"rectangle"` representing a rectangle.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`owin`, `as.owin`, `boundingbox`

Examples

```
w <- owin(c(0,10),c(0,10), poly=list(x=c(1,2,3,2,1), y=c(2,3,4,6,7)))
r <- as.rectangle(w)
# returns a 10 x 10 rectangle

data(lansing)
as.rectangle(lansing)

data(copper)
as.rectangle(copper$SouthLines)
```

---

as.solist — *Convert List of Two-Dimensional Spatial Objects*

Description

Given a list of two-dimensional spatial objects, convert it to the class `"solist"`.

Usage

```
as.solist(x, ...)
```

Arguments

`x`	A list of objects, each representing a two-dimensional spatial dataset.
`...`	Additional arguments passed to `solist`.

Details

This command makes the list x into an object of class "solist" (spatial object list). See solist for details.

The entries in the list x should be two-dimensional spatial datasets (not necessarily of the same class).

Value

A list, usually of class "solist".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

solist, as.anylist, solapply.

Examples

```
x <- list(cells, density(cells))
y <- as.solist(x)
```

---

as.tess *Convert Data To Tessellation*

Description

Converts data specifying a tessellation, in any of several formats, into an object of class "tess".

Usage

```
 as.tess(X)
 ## S3 method for class 'tess'
as.tess(X)
 ## S3 method for class 'im'
as.tess(X)
 ## S3 method for class 'owin'
as.tess(X)
 ## S3 method for class 'quadratcount'
as.tess(X)
 ## S3 method for class 'quadrattest'
as.tess(X)
 ## S3 method for class 'list'
as.tess(X)
```

Arguments

X	Data to be converted to a tessellation.

Details

A tessellation is a collection of disjoint spatial regions (called *tiles*) that fit together to form a larger spatial region. This command creates an object of class "tess" that represents a tessellation.

This function converts data in any of several formats into an object of class "tess" for use by the **spatstat** package. The argument X may be

- an object of class "tess". The object will be stripped of any extraneous attributes and returned.
- a pixel image (object of class "im") with pixel values that are logical or factor values. Each level of the factor will determine a tile of the tessellation.
- a window (object of class "owin"). The result will be a tessellation consisting of a single tile.
- a set of quadrat counts (object of class "quadratcount") returned by the command quadratcount. The quadrats used to generate the counts will be extracted and returned as a tessellation.
- a quadrat test (object of class "quadrattest") returned by the command quadrat.test. The quadrats used to perform the test will be extracted and returned as a tessellation.
- a list of windows (objects of class "owin") giving the tiles of the tessellation.

The function as.tess is generic, with methods for various classes, as listed above.

Value

An object of class "tess" specifying a tessellation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

tess

Examples

```
# pixel image
v <- as.im(function(x,y){factor(round(5 * (x^2 + y^2)))}, W=owin())
levels(v) <- letters[seq(length(levels(v)))]
as.tess(v)
# quadrat counts
data(nztrees)
qNZ <- quadratcount(nztrees, nx=4, ny=3)
as.tess(qNZ)
```

auc *Area Under ROC Curve*

Description

Compute the AUC (area under the Receiver Operating Characteristic curve) for a fitted point process model.

Usage

```
auc(X, ...)

## S3 method for class 'ppp'
auc(X, covariate, ..., high = TRUE)

## S3 method for class 'ppm'
auc(X, ...)

## S3 method for class 'kppm'
auc(X, ...)

## S3 method for class 'lpp'
auc(X, covariate, ..., high = TRUE)

## S3 method for class 'lppm'
auc(X, ...)
```

Arguments

X	Point pattern (object of class "ppp" or "lpp") or fitted point process model (object of class "ppm" or "kppm" or "lppm").
covariate	Spatial covariate. Either a `function(x,y)`, a pixel image (object of class "im"), or one of the strings "x" or "y" indicating the Cartesian coordinates.
...	Arguments passed to `as.mask` controlling the pixel resolution for calculations.
high	Logical value indicating whether the threshold operation should favour high or low values of the covariate.

Details

This command computes the AUC, the area under the Receiver Operating Characteristic curve. The ROC itself is computed by `roc`.

For a point pattern `X` and a covariate `Z`, the AUC is a numerical index that measures the ability of the covariate to separate the spatial domain into areas of high and low density of points. Let x_i be a randomly-chosen data point from `X` and U a randomly-selected location in the study region. The AUC is the probability that $Z(x_i) > Z(U)$ assuming `high=TRUE`. That is, AUC is the probability that a randomly-selected data point has a higher value of the covariate `Z` than does a randomly-selected spatial location. The AUC is a number between 0 and 1. A value of 0.5 indicates a complete lack of discriminatory power.

For a fitted point process model `X`, the AUC measures the ability of the fitted model intensity to separate the spatial domain into areas of high and low density of points. Suppose $\lambda(u)$ is the intensity function of the model. The AUC is the probability that $\lambda(x_i) > \lambda(U)$. That is, AUC is the probability that a randomly-selected data point has higher predicted intensity than does a randomly-selected spatial location. The AUC is **not** a measure of the goodness-of-fit of the model (Lobo et al, 2007).

Value

Numeric. For `auc.ppp` and `auc.lpp`, the result is a single number giving the AUC value. For `auc.ppm`, `auc.kppm` and `auc.lppm`, the result is a numeric vector of length 2 giving the AUC value and the theoretically expected AUC value for this model.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Lobo, J.M., Jiménez-Valverde, A. and Real, R. (2007) AUC: a misleading measure of the performance of predictive distribution models. *Global Ecology and Biogeography* **17**(2) 145–151.

Nam, B.-H. and D'Agostino, R. (2002) Discrimination index, the area under the ROC curve. Pages 267–279 in Huber-Carol, C., Balakrishnan, N., Nikulin, M.S. and Mesbah, M., *Goodness-of-fit tests and model validity*, Birkhäuser, Basel.

See Also

`roc`

Examples

```
fit <- ppm(swedishpines ~ x+y)
auc(fit)
auc(swedishpines, "x")
```

`BadGey` *Hybrid Geyer Point Process Model*

Description

Creates an instance of the Baddeley-Geyer point process model, defined as a hybrid of several Geyer interactions. The model can then be fitted to point pattern data.

Usage

```
BadGey(r, sat)
```

Arguments

`r`	vector of interaction radii
`sat`	vector of saturation parameters, or a single common value of saturation parameter

Details

This is Baddeley's generalisation of the Geyer saturation point process model, described in Geyer, to a process with multiple interaction distances.

The BadGey point process with interaction radii $r_1, \ldots, r_k$, saturation thresholds $s_1, \ldots, s_k$, intensity parameter β and interaction parameters $\gamma_1, \ldots, gamma_k$, is the point process in which each point x_i in the pattern X contributes a factor

$$\beta \gamma_1^{v_1(x_i, X)} \ldots gamma_k^{v_k(x_i, X)}$$

to the probability density of the point pattern, where

$$v_j(x_i, X) = \min(s_j, t_j(x_i, X))$$

where $t_j(x_i, X)$ denotes the number of points in the pattern X which lie within a distance r_j from the point x_i.

`BadGey` is used to fit this model to data. The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the piecewise constant Saturated pairwise interaction is yielded by the function `BadGey()`. See the examples below.

The argument `r` specifies the vector of interaction distances. The entries of `r` must be strictly increasing, positive numbers.

The argument `sat` specifies the vector of saturation parameters that are applied to the point counts $t_j(x_i, X)$. It should be a vector of the same length as `r`, and its entries should be nonnegative numbers. Thus `sat[1]` is applied to the count of points within a distance `r[1]`, and `sat[2]` to the count of points within a distance `r[2]`, etc. Alternatively `sat` may be a single number, and this saturation value will be applied to every count.

Infinite values of the saturation parameters are also permitted; in this case $v_j(x_i, X) = t_j(x_i, X)$ and there is effectively no 'saturation' for the distance range in question. If all the saturation parameters are set to `Inf` then the model is effectively a pairwise interaction process, equivalent to `PairPiece` (however the interaction parameters γ obtained from `BadGey` have a complicated relationship to the interaction parameters γ obtained from `PairPiece`).

If `r` is a single number, this model is virtually equivalent to the Geyer process, see `Geyer`.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz> in collaboration with Hao Wang and Jeff Picka

See Also

`ppm`, `pairsat.family`, `Geyer`, `PairPiece`, `SatPiece`

Examples

```
BadGey(c(0.1,0.2), c(1,1))
# prints a sensible description of itself
BadGey(c(0.1,0.2), 1)
data(cells)
```

```
    # fit a stationary Baddeley-Geyer model
    ppm(cells, ~1, BadGey(c(0.07, 0.1, 0.13), 2))

    # nonstationary process with log-cubic polynomial trend
    ## Not run:
    ppm(cells, ~polynom(x,y,3), BadGey(c(0.07, 0.1, 0.13), 2))

## End(Not run)
```

---

bc.ppm *Bias Correction for Fitted Model*

Description

Applies a first-order bias correction to a fitted model.

Usage

```
  bc(fit, ...)

  ## S3 method for class 'ppm'
bc(fit, ..., nfine = 256)
```

Arguments

fit	A fitted point process model (object of class "ppm") or a model of some other class.
...	Additional arguments are currently ignored.
nfine	Grid dimensions for fine grid of locations. An integer, or a pair of integers. See Details.

Details

This command applies the first order Newton-Raphson bias correction method of Baddeley and Turner (2014, sec 4.2) to a fitted model. The function bc is generic, with a method for fitted point process models of class "ppm".

A fine grid of locations, of dimensions nfine * nfine or nfine[2] * nfine[1], is created over the original window of the data, and the intensity or conditional intensity of the fitted model is calculated on this grid. The result is used to update the fitted model parameters once by a Newton-Raphson update.

This is only useful if the quadrature points used to fit the original model fit are coarser than the grid of points specified by nfine.

Value

A numeric vector, of the same length as coef(fit), giving updated values for the fitted model coefficients.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

References

Baddeley, A. and Turner, R. (2014) Bias correction for parameter estimates of spatial point process models. *Journal of Statistical Computation and Simulation* **84**, 1621–1643. DOI: 10.1080/00949655.2012.755976

See Also

rex

Examples

```
fit <- ppm(cells ~ x, Strauss(0.07))
coef(fit)
if(!interactive()) {
  bc(fit, nfine=64)
} else {
  bc(fit)
}
```

---

bdist.pixels *Distance to Boundary of Window*

Description

Computes the distances from each pixel in a window to the boundary of the window.

Usage

```
bdist.pixels(w, ..., style="image", method=c("C", "interpreted"))
```

Arguments

`w`	A window (object of class "owin").
`...`	Arguments passed to `as.mask` to determine the pixel resolution.
`style`	Character string determining the format of the output: either "matrix", "coords" or "image".
`method`	Choice of algorithm to use when `w` is polygonal.

Details

This function computes, for each pixel u in the window `w`, the shortest distance $d(u, W^c)$ from u to the boundary of W.

If the window is a binary mask then the distance from each pixel to the boundary is computed using the distance transform algorithm `distmap.owin`. The result is equivalent to `distmap(W, invert=TRUE)`.

If the window is a rectangle or a polygonal region, the grid of pixels is determined by the arguments "..." passed to `as.mask`. The distance from each pixel to the boundary is calculated exactly, using analytic geometry. This is slower but more accurate than in the case of a binary mask.

For software testing purposes, there are two implementations available when `w` is a polygon: the default is `method="C"` which is much faster than `method="interpreted"`.

Value

If `style="image"`, a pixel image (object of class `"im"`) containing the distances from each pixel in the image raster to the boundary of the window.

If `style="matrix"`, a matrix giving the distances from each pixel in the image raster to the boundary of the window. Rows of this matrix correspond to the y coordinate and columns to the x coordinate.

If `style="coords"`, a list with three components `x,y,z`, where `x,y` are vectors of length m, n giving the x and y coordinates respectively, and `z` is an $m \times n$ matrix such that `z[i,j]` is the distance from `(x[i],y[j])` to the boundary of the window. Rows of this matrix correspond to the x coordinate and columns to the y coordinate. This result can be plotted with `persp`, `image` or `contour`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`owin.object`, `erosion`, `bdist.points`, `bdist.tiles`, `distmap.owin`.

Examples

```
u <- owin(c(0,1),c(0,1))
d <- bdist.pixels(u, eps=0.01)
image(d)
d <- bdist.pixels(u, eps=0.01, style="matrix")
mean(d >= 0.1)
# value is approx (1 - 2 * 0.1)^2 = 0.64
```

`bdist.points` *Distance to Boundary of Window*

Description

Computes the distances from each point of a point pattern to the boundary of the window.

Usage

```
bdist.points(X)
```

Arguments

`X`	A point pattern (object of class `"ppp"`).

Details

This function computes, for each point x_i in the point pattern `X`, the shortest distance $d(x_i, W^c)$ from x_i to the boundary of the window W of observation.

If the window `Window(X)` is of type `"rectangle"` or `"polygonal"`, then these distances are computed by analytic geometry and are exact, up to rounding errors. If the window is of type `"mask"` then the distances are computed using the real-valued distance transform, which is an approximation with maximum error equal to the width of one pixel in the mask.

Value

A numeric vector, giving the distances from each point of the pattern to the boundary of the window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

bdist.pixels, bdist.tiles, ppp.object, erosion

Examples

```
data(cells)
d <- bdist.points(cells)
```

---

bdist.tiles *Distance to Boundary of Window*

Description

Computes the shortest distances from each tile in a tessellation to the boundary of the window.

Usage

```
bdist.tiles(X)
```

Arguments

X	A tessellation (object of class "tess").

Details

This function computes, for each tile s_i in the tessellation X, the shortest distance from s_i to the boundary of the window W containing the tessellation.

Value

A numeric vector, giving the shortest distance from each tile in the tessellation to the boundary of the window. Entries of the vector correspond to the entries of tiles(X).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

tess, bdist.points, bdist.pixels

Examples

```
P <- runifpoint(15)
X <- dirichlet(P)
plot(X, col="red")
B <- bdist.tiles(X)
# identify tiles that do not touch the boundary
plot(X[B > 0], add=TRUE, col="green", lwd=3)
```

---

beachcolours *Create Colour Scheme for a Range of Numbers*

Description

Given a range of numerical values, this command creates a colour scheme that would be appropriate if the numbers were altitudes (elevation above or below sea level).

Usage

```
beachcolours(range, sealevel = 0, monochrome = FALSE,
             ncolours = if (monochrome) 16 else 64,
             nbeach = 1)
beachcolourmap(range, ...)
```

Arguments

range	Range of numerical values to be mapped. A numeric vector of length 2.
sealevel	Value that should be treated as zero. A single number, lying between range[1] and range[2].
monochrome	Logical. If TRUE then a greyscale colour map is constructed.
ncolours	Number of distinct colours to use.
nbeach	Number of colours that will be yellow.
...	Arguments passed to beachcolours.

Details

Given a range of numerical values, these commands create a colour scheme that would be appropriate if the numbers were altitudes (elevation above or below sea level).

Numerical values close to zero are portrayed in green (representing the waterline). Negative values are blue (representing water) and positive values are yellow to red (representing land). At least, these are the colours of land and sea in Western Australia. This colour scheme was proposed by Baddeley et al (2005).

The function beachcolours returns these colours as a character vector, while beachcolourmap returns a colourmap object.

The argument range should be a numeric vector of length 2 giving a range of numerical values.

The argument sealevel specifies the height value that will be treated as zero, and mapped to the colour green. A vector of ncolours colours will be created, of which nbeach colours will be green.

The argument monochrome is included for convenience when preparing publications. If monochrome=TRUE the colour map will be a simple grey scale containing ncolours shades from black to white.

Value

For beachcolours, a character vector of length ncolours specifying colour values. For beachcolourmap, a colour map (object of class "colourmap").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

See Also

colourmap, colourtools.

Examples

```
plot(beachcolourmap(c(-2,2)))
```

---

beginner *Print Introduction For Beginners*

Description

Prints an introduction for beginners to the spatstat package, or another specified package.

Usage

```
beginner(package = "spatstat")
```

Arguments

package	Name of package.

Details

This function prints an introduction for beginners to the **spatstat** package.

The function can be executed simply by typing beginner without parentheses.

If the argument package is given, then the function prints the beginner's help file BEGINNER.txt from the specified package (if it has one).

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

latest.news

Examples

```
beginner
```

---

begins *Check Start of Character String*

Description

Checks whether a character string begins with a particular prefix.

Usage

```
begins(x, firstbit)
```

Arguments

x	Character string, or vector of character strings, to be tested.
firstbit	A single character string.

Details

This simple wrapper function checks whether (each entry in) x begins with the string firstbit, and returns a logical value or logical vector with one entry for each entry of x. This function is useful mainly for reducing complexity in model formulae.

Value

Logical vector of the same length as x.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
begins(c("Hello", "Goodbye"), "Hell")
begins("anything", "")
```

berman.test *Berman's Tests for Point Process Model*

Description

Tests the goodness-of-fit of a Poisson point process model using methods of Berman (1986).

Usage

```
berman.test(...)

## S3 method for class 'ppp'
berman.test(X, covariate,
                         which = c("Z1", "Z2"),
        alternative = c("two.sided", "less", "greater"), ...)

## S3 method for class 'ppm'
berman.test(model, covariate,
                         which = c("Z1", "Z2"),
               alternative = c("two.sided", "less", "greater"), ...)

## S3 method for class 'lpp'
berman.test(X, covariate,
                         which = c("Z1", "Z2"),
        alternative = c("two.sided", "less", "greater"), ...)

## S3 method for class 'lppm'
berman.test(model, covariate,
                         which = c("Z1", "Z2"),
               alternative = c("two.sided", "less", "greater"), ...)
```

Arguments

`X`	A point pattern (object of class `"ppp"` or `"lpp"`).
`model`	A fitted point process model (object of class `"ppm"` or `"lppm"`).
`covariate`	The spatial covariate on which the test will be based. An image (object of class `"im"`) or a function.
`which`	Character string specifying the choice of test.
`alternative`	Character string specifying the alternative hypothesis.
`...`	Additional arguments controlling the pixel resolution (arguments `dimyx` and `eps` passed to `as.mask`) or other undocumented features.

Details

These functions perform a goodness-of-fit test of a Poisson point process model fitted to point pattern data. The observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same values under the model, are compared using either of two test statistics Z_1 and Z_2 proposed by Berman (1986). The Z_1 test is also known as the Lawson-Waller test.

The function berman.test is generic, with methods for point patterns ("ppp" or "lpp") and point process models ("ppm" or "lppm").

- If X is a point pattern dataset (object of class "ppp" or "lpp"), then berman.test(X, ...) performs a goodness-of-fit test of the uniform Poisson point process (Complete Spatial Randomness, CSR) for this dataset.
- If model is a fitted point process model (object of class "ppm" or "lppm") then berman.test(model, ...) performs a test of goodness-of-fit for this fitted model. In this case, model should be a Poisson point process.

The test is performed by comparing the observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same covariate under the model. Thus, you must nominate a spatial covariate for this test.

The argument covariate should be either a function(x,y) or a pixel image (object of class "im" containing the values of a spatial function. If covariate is an image, it should have numeric values, and its domain should cover the observation window of the model. If covariate is a function, it should expect two arguments x and y which are vectors of coordinates, and it should return a numeric vector of the same length as x and y.

First the original data point pattern is extracted from model. The values of the covariate at these data points are collected.

Next the values of the covariate at all locations in the observation window are evaluated. The point process intensity of the fitted model is also evaluated at all locations in the window.

- If which="Z1", the test statistic Z_1 is computed as follows. The sum S of the covariate values at all data points is evaluated. The predicted mean μ and variance σ^2 of S are computed from the values of the covariate at all locations in the window. Then we compute $Z_1 = (S - \mu)/\sigma$. Closely-related tests were proposed independently by Waller et al (1993) and Lawson (1993) so this test is often termed the Lawson-Waller test in epidemiological literature.
- If which="Z2", the test statistic Z_2 is computed as follows. The values of the covariate at all locations in the observation window, weighted by the point process intensity, are compiled into a cumulative distribution function F. The probability integral transformation is then applied: the values of the covariate at the original data points are transformed by the predicted cumulative distribution function F into numbers between 0 and 1. If the model is correct, these numbers are i.i.d. uniform random numbers. The standardised sample mean of these numbers is the statistic Z_2.

In both cases the null distribution of the test statistic is the standard normal distribution, approximately.

The return value is an object of class "htest" containing the results of the hypothesis test. The print method for this class gives an informative summary of the test outcome.

Value

An object of class "htest" (hypothesis test) and also of class "bermantest", containing the results of the test. The return value can be plotted (by plot.bermantest) or printed to give an informative summary of the test.

Warning

The meaning of a one-sided test must be carefully scrutinised: see the printed output.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

References

Berman, M. (1986) Testing for spatial association between a point process and another stochastic process. *Applied Statistics* **35**, 54–62.

Lawson, A.B. (1993) On the analysis of mortality events around a prespecified fixed point. *Journal of the Royal Statistical Society, Series A* **156** (3) 363–377.

Waller, L., Turnbull, B., Clark, L.C. and Nasca, P. (1992) Chronic Disease Surveillance and testing of clustering of disease and exposure: Application to leukaemia incidence and TCE-contaminated dumpsites in upstate New York. *Environmetrics* **3**, 281–300.

See Also

cdf.test, quadrat.test, ppm

Examples

```
# Berman's data
data(copper)
X <- copper$SouthPoints
L <- copper$SouthLines
D <- distmap(L, eps=1)
# test of CSR
berman.test(X, D)
berman.test(X, D, "Z2")
```

---

bind.fv *Combine Function Value Tables*

Description

Advanced Use Only. Combine objects of class "fv", or glue extra columns of data onto an existing "fv" object.

Usage

```
## S3 method for class 'fv'
cbind(...)
bind.fv(x, y, labl = NULL, desc = NULL, preferred = NULL, clip=FALSE)
```

Arguments

...	Any number of arguments, which are objects of class "fv".
x	An object of class "fv".
y	Either a data frame or an object of class "fv".
labl	Plot labels (see fv) for columns of y. A character vector.

desc	Descriptions (see fv) for columns of y. A character vector.
preferred	Character string specifying the column which is to be the new recommended value of the function.
clip	Logical value indicating whether each object must have exactly the same domain, that is, the same sequence of values of the function argument (clip=FALSE, the default) or whether objects with different domains are permissible and will be restricted to a common domain (clip=TRUE).

Details

This documentation is provided for experienced programmers who want to modify the internal behaviour of **spatstat**.

The function cbind.fv is a method for the generic R function cbind. It combines any number of objects of class "fv" into a single object of class "fv". The objects must be compatible, in the sense that they have identical values of the function argument.

The function bind.fv is a lower level utility which glues additional columns onto an existing object x of class "fv". It has two modes of use:

- If the additional dataset y is an object of class "fv", then x and y must be compatible as described above. Then the columns of y that contain function values will be appended to the object x.
- Alternatively if y is a data frame, then y must have the same number of rows as x. All columns of y will be appended to x.

The arguments labl and desc provide plot labels and description strings (as described in fv) for the *new* columns. If y is an object of class "fv" then labl and desc are optional, and default to the relevant entries in the object y. If y is a data frame then labl and desc must be provided.

Value

An object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

fv, with.fv.

Undocumented functions for modifying an "fv" object include fvnames, fvnames<-, tweak.fv.entry and rebadge.fv.

Examples

```
data(cells)
K1 <- Kest(cells, correction="border")
K2 <- Kest(cells, correction="iso")
# remove column 'theo' to avoid duplication
K2 <- K2[, names(K2) != "theo"]

cbind(K1, K2)
```

```
bind.fv(K1, K2, preferred="iso")

# constrain border estimate to be monotonically increasing
bm <- cumsum(c(0, pmax(0, diff(K1$border))))
bind.fv(K1, data.frame(bmono=bm),
            "%s[bmo](r)",
            "monotone border-corrected estimate of %s",
            "bmono")
```

---

bits.test *Balanced Independent Two-Stage Monte Carlo Test*

Description

Performs a Balanced Independent Two-Stage Monte Carlo test of goodness-of-fit for spatial pattern.

Usage

```
bits.test(X, ...,
         exponent = 2, nsim=19,
         alternative=c("two.sided", "less", "greater"),
         leaveout=1, interpolate = FALSE,
         savefuns=FALSE, savepatterns=FALSE,
         verbose = TRUE)
```

Arguments

X	Either a point pattern dataset (object of class "ppp", "lpp" or "pp3") or a fitted point process model (object of class "ppm", "kppm", "lppm" or "slrm").
...	Arguments passed to dclf.test or mad.test or envelope to control the conduct of the test. Useful arguments include fun to determine the summary function, rinterval to determine the range of r values used in the test, and use.theory described under Details.
exponent	Exponent used in the test statistic. Use exponent=2 for the Diggle-Cressie-Loosmore-Ford test, and exponent=Inf for the Maximum Absolute Deviation test.
nsim	Number of replicates in each stage of the test. A total of nsim * (nsim + 1) simulated point patterns will be generated, and the p-value will be a multiple of 1/(nsim+1).
alternative	Character string specifying the alternative hypothesis. The default (alternative="two.sided") is that the true value of the summary function is not equal to the theoretical value postulated under the null hypothesis. If alternative="less" the alternative hypothesis is that the true value of the summary function is lower than the theoretical value.
leaveout	Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.
interpolate	Logical value indicating whether to interpolate the distribution of the test statistic by kernel smoothing, as described in Dao and Genton (2014, Section 5).

savefuns	Logical flag indicating whether to save the simulated function values (from the first stage).
savepatterns	Logical flag indicating whether to save the simulated point patterns (from the first stage).
verbose	Logical value indicating whether to print progress reports.

Details

Performs the Balanced Independent Two-Stage Monte Carlo test proposed by Baddeley et al (2017), an improvement of the Dao-Genton (2014) test.

If X is a point pattern, the null hypothesis is CSR.

If X is a fitted model, the null hypothesis is that model.

The argument `use.theory` passed to `envelope` determines whether to compare the summary function for the data to its theoretical value for CSR (`use.theory=TRUE`) or to the sample mean of simulations from CSR (`use.theory=FALSE`).

The argument `leaveout` specifies how to calculate the discrepancy between the summary function for the data and the nominal reference value, when the reference value must be estimated by simulation. The values `leaveout=0` and `leaveout=1` are both algebraically equivalent (Baddeley et al, 2014, Appendix) to computing the difference `observed - reference` where the `reference` is the mean of simulated values. The value `leaveout=2` gives the leave-two-out discrepancy proposed by Dao and Genton (2014).

Value

A hypothesis test (object of class `"htest"` which can be printed to show the outcome of the test.

Author(s)

Adrian Baddeley, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.n and Ege Rubak <rubak@math.aau.dk>.

References

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

Baddeley, A., Diggle, P.J., Hardegen, A., Lawrence, T., Milne, R.K. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84** (3) 477–489.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2017) On two-stage Monte Carlo tests of composite hypotheses. *Computational Statistics and Data Analysis*, in press.

See Also

`dg.test`, `dclf.test`, `mad.test`

Examples

```
ns <- if(interactive()) 19 else 4
bits.test(cells, nsim=ns)
bits.test(cells, alternative="less", nsim=ns)
bits.test(cells, nsim=ns, interpolate=TRUE)
```

`blur` *Apply Gaussian Blur to a Pixel Image*

Description

Applies a Gaussian blur to a pixel image.

Usage

```
blur(x, sigma = NULL, ..., normalise=FALSE, bleed = TRUE, varcov=NULL)

## S3 method for class 'im'
Smooth(X, sigma = NULL, ...,
                    normalise=FALSE, bleed = TRUE, varcov=NULL)
```

Arguments

`x,X`	The pixel image. An object of class `"im"`.
`sigma`	Standard deviation of isotropic Gaussian smoothing kernel.
`...`	Ignored.
`normalise`	Logical flag indicating whether the output values should be divided by the corresponding blurred image of the window itself. See Details.
`bleed`	Logical flag indicating whether to allow blur to extend outside the original domain of the image. See Details.
`varcov`	Variance-covariance matrix of anisotropic Gaussian kernel. Incompatible with `sigma`.

Details

This command applies a Gaussian blur to the pixel image `x`.

`Smooth.im` is a method for the generic `Smooth` for pixel images. It is currently identical to `blur`, apart from the name of the first argument.

The blurring kernel is the isotropic Gaussian kernel with standard deviation `sigma`, or the anisotropic Gaussian kernel with variance-covariance matrix `varcov`. The arguments `sigma` and `varcov` are incompatible. Also `sigma` may be a vector of length 2 giving the standard deviations of two independent Gaussian coordinates, thus equivalent to `varcov = diag(sigma^2)`.

If the pixel values of `x` include some `NA` values (meaning that the image domain does not completely fill the rectangular frame) then these `NA` values are first reset to zero.

The algorithm then computes the convolution $x * G$ of the (zero-padded) pixel image x with the specified Gaussian kernel G.

If `normalise=FALSE`, then this convolution $x * G$ is returned. If `normalise=TRUE`, then the convolution $x * G$ is normalised by dividing it by the convolution $w * G$ of the image domain `w` with the same Gaussian kernel. Normalisation ensures that the result can be interpreted as a weighted average of input pixel values, without edge effects due to the shape of the domain.

If `bleed=FALSE`, then pixel values outside the original image domain are set to `NA`. Thus the output is a pixel image with the same domain as the input. If `bleed=TRUE`, then no such alteration is performed, and the result is a pixel image defined everywhere in the rectangular frame containing the input image.

Computation is performed using the Fast Fourier Transform.

Value

A pixel image with the same pixel array as the input image `x`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`interp.im` for interpolating a pixel image to a finer resolution, `density.ppp` for blurring a point pattern, `Smooth.ppp` for interpolating marks attached to points.

Examples

```
data(letterR)
Z <- as.im(function(x,y) { 4 * x^2 + 3 * y }, letterR)
par(mfrow=c(1,3))
plot(Z)
plot(letterR, add=TRUE)
plot(blur(Z, 0.3, bleed=TRUE))
plot(letterR, add=TRUE)
plot(blur(Z, 0.3, bleed=FALSE))
plot(letterR, add=TRUE)
par(mfrow=c(1,1))
```

---

border — *Border Region of a Window*

Description

Computes the border region of a window, that is, the region lying within a specified distance of the boundary of a window.

Usage

```
border(w, r, outside=FALSE, ...)
```

Arguments

`w`	A window (object of class `"owin"`) or something acceptable to `as.owin`.
`r`	Numerical value.
`outside`	Logical value determining whether to compute the border outside or inside `w`.
`...`	Optional arguments passed to `erosion` (if `outside=FALSE`) or to `dilation` (if `outside=TRUE`).

Details

By default (if `outside=FALSE`), the border region is the subset of `w` lying within a distance `r` of the boundary of `w`. It is computed by eroding `w` by the distance `r` (using `erosion`) and subtracting this eroded window from the original window `w`.

If `outside=TRUE`, the border region is the set of locations outside `w` lying within a distance `r` of `w`. It is computed by dilating `w` by the distance `r` (using `dilation`) and subtracting the original window `w` from the dilated window.

Value

A window (object of class `"owin"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`erosion`, `dilation`

Examples

```
# rectangle
   u <- unit.square()
   border(u, 0.1)
   border(u, 0.1, outside=TRUE)
# polygon

   data(letterR)
   plot(letterR)
   plot(border(letterR, 0.1), add=TRUE)
   plot(border(letterR, 0.1, outside=TRUE), add=TRUE)
```

`bounding.box.xy` *Convex Hull of Points*

Description

Computes the smallest rectangle containing a set of points.

Usage

```
bounding.box.xy(x, y=NULL)
```

Arguments

`x`	vector of `x` coordinates of observed points, or a 2-column matrix giving `x,y` coordinates, or a list with components `x,y` giving coordinates (such as a point pattern object of class `"ppp"`.)
`y`	(optional) vector of `y` coordinates of observed points, if `x` is a vector.

Details

Given an observed pattern of points with coordinates given by `x` and `y`, this function finds the smallest rectangle, with sides parallel to the coordinate axes, that contains all the points, and returns it as a window.

Value

A window (an object of class `"owin"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`owin`, `as.owin`, `convexhull.xy`, `ripras`

Examples

```
x <- runif(30)
y <- runif(30)
w <- bounding.box.xy(x,y)
plot(owin(), main="bounding.box.xy(x,y)")
plot(w, add=TRUE)
points(x,y)

X <- rpoispp(30)
plot(X, main="bounding.box.xy(X)")
plot(bounding.box.xy(X), add=TRUE)
```

boundingbox *Bounding Box of a Window, Image, or Point Pattern*

Description

Find the smallest rectangle containing a given window(s), image(s) or point pattern(s).

Usage

```
boundingbox(...)

## Default S3 method:
boundingbox(...)

## S3 method for class 'im'
boundingbox(...)

## S3 method for class 'owin'
boundingbox(...)

## S3 method for class 'ppp'
boundingbox(...)

## S3 method for class 'psp'
boundingbox(...)

## S3 method for class 'lpp'
boundingbox(...)

## S3 method for class 'linnet'
boundingbox(...)

## S3 method for class 'solist'
boundingbox(...)
```

Arguments

...	One or more windows (objects of class "owin"), pixel images (objects of class "im") or point patterns (objects of class "ppp" or "lpp") or line segment patterns (objects of class "psp") or linear networks (objects of class "linnet") or any combination of such objects. Alternatively, the argument may be a list of such objects, of class "solist".

Details

This function finds the smallest rectangle (with sides parallel to the coordinate axes) that contains all the given objects.

For a window (object of class "owin"), the bounding box is the smallest rectangle that contains all the vertices of the window (this is generally smaller than the enclosing frame, which is returned by as.rectangle).

For a point pattern (object of class "ppp" or "lpp"), the bounding box is the smallest rectangle that contains all the points of the pattern. This is usually smaller than the bounding box of the window of the point pattern.

For a line segment pattern (object of class "psp") or a linear network (object of class "linnet"), the bounding box is the smallest rectangle that contains all endpoints of line segments.

For a pixel image (object of class "im"), the image will be converted to a window using as.owin, and the bounding box of this window is obtained.

If the argument is a list of several objects, then this function finds the smallest rectangle that contains all the bounding boxes of the objects.

Value

owin, as.owin, as.rectangle

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

Examples

```
w <- owin(c(0,10),c(0,10), poly=list(x=c(1,2,3,2,1), y=c(2,3,4,6,7)))
r <- boundingbox(w)
# returns rectangle [1,3] x [2,7]

w2 <- unit.square()
r <- boundingbox(w, w2)
# returns rectangle [0,3] x [0,7]
```

---

boundingcircle *Smallest Enclosing Circle*

Description

Find the smallest circle enclosing a spatial window or other object. Return its radius, or the location of its centre, or the circle itself.

Usage

```
boundingradius(x, ...)

boundingcentre(x, ...)

boundingcircle(x, ...)

## S3 method for class 'owin'
boundingradius(x, ...)

## S3 method for class 'owin'
boundingcentre(x, ...)
```

```
## S3 method for class 'owin'
boundingcircle(x, ...)

## S3 method for class 'ppp'
boundingradius(x, ...)

## S3 method for class 'ppp'
boundingcentre(x, ...)

## S3 method for class 'ppp'
boundingcircle(x, ...)
```

Arguments

x	A window (object of class "owin"), or another spatial object.
...	Arguments passed to as.mask to determine the pixel resolution for the calculation.

Details

The boundingcircle of a spatial region W is the smallest circle that contains W. The boundingradius is the radius of this circle, and the boundingcentre is the centre of the circle.

The functions boundingcircle, boundingcentre and boundingradius are generic. There are methods for objects of class "owin", "ppp" and "linnet".

Value

The result of boundingradius is a single numeric value.

The result of boundingcentre is a point pattern containing a single point.

The result of boundingcircle is a window representing the boundingcircle.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

boundingradius.linnet

Examples

```
boundingradius(letterR)

plot(grow.rectangle(Frame(letterR), 0.2), main="", type="n")
plot(letterR, add=TRUE, col="grey")
plot(boundingcircle(letterR), add=TRUE, border="green", lwd=2)
plot(boundingcentre(letterR), pch="+", cex=2, col="blue", add=TRUE)

X <- runifpoint(5)
plot(X)
plot(boundingcircle(X), add=TRUE)
plot(boundingcentre(X), pch="+", cex=2, col="blue", add=TRUE)
```

box3 *Three-Dimensional Box*

Description

Creates an object representing a three-dimensional box.

Usage

```
box3(xrange = c(0, 1), yrange = xrange, zrange = yrange, unitname = NULL)
```

Arguments

`xrange, yrange, zrange`
: Dimensions of the box in the x, y, z directions. Each of these arguments should be a numeric vector of length 2.

`unitname`
: Optional. Name of the unit of length. See Details.

Details

This function creates an object representing a three-dimensional rectangular parallelepiped (box) with sides parallel to the coordinate axes.

The object can be used to specify the domain of a three-dimensional point pattern (see `pp3`) and in various geometrical calculations (see `volume.box3`, `diameter.box3`, `eroded.volumes`).

The optional argument `unitname` specifies the name of the unit of length. See `unitname` for valid formats.

The function `as.box3` can be used to convert other kinds of data to this format.

Value

An object of class `"box3"`. There is a print method for this class.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`as.box3`, `pp3`, `volume.box3`, `diameter.box3`, `eroded.volumes`.

Examples

```
box3()
box3(c(0,10),c(0,10),c(0,5), unitname=c("metre","metres"))
box3(c(-1,1))
```

boxx *Multi-Dimensional Box*

Description

Creates an object representing a multi-dimensional box.

Usage

```
boxx(..., unitname = NULL)
```

Arguments

...	Dimensions of the box. Vectors of length 2.
unitname	Optional. Name of the unit of length. See Details.

Details

This function creates an object representing a multi-dimensional rectangular parallelepiped (box) with sides parallel to the coordinate axes.

The object can be used to specify the domain of a multi-dimensional point pattern (see ppx) and in various geometrical calculations (see volume.boxx, diameter.boxx, eroded.volumes).

The optional argument unitname specifies the name of the unit of length. See unitname for valid formats.

Value

An object of class "boxx". There is a print method for this class.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

ppx, volume.boxx, diameter.boxx, eroded.volumes.boxx.

Examples

```
boxx(c(0,10),c(0,10),c(0,5),c(0,1), unitname=c("metre","metres"))
```

branchlabelfun *Tree Branch Membership Labelling Function*

Description

Creates a function which returns the tree branch membership label for any location on a linear network.

Usage

```
branchlabelfun(L, root = 1)
```

Arguments

`L`	Linear network (object of class `"linnet"`). The network must have no loops.
`root`	Root of the tree. An integer index identifying which point in `vertices(L)` is the root of the tree.

Details

The linear network `L` must be an acyclic graph (i.e. must not contain any loops) so that it can be interpreted as a tree.

The result of `f <- branchlabelfun(L, root)` is a function `f` which gives, for each location on the linear network `L`, the tree branch label at that location.

Tree branch labels are explained in `treebranchlabels`.

The result `f` also belongs to the class `"linfun"`. It can be called using several different kinds of data, as explained in the help for `linfun`. The values of the function are character strings.

Value

A function (of class `"linfun"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`treebranchlabels`, `linfun`

Examples

```
# make a simple tree
m <- simplenet$m
m[8,10] <- m[10,8] <- FALSE
L <- linnet(vertices(simplenet), m)
# make function
f <- branchlabelfun(L, 1)
plot(f)
```

```
X <- runiflpp(5, L)
f(X)
```

---

bugfixes *List Recent Bug Fixes*

Description

List all bug fixes in a package, starting from a certain date or version of the package. Fixes are sorted alphabetically by the name of the affected function. The default is to list bug fixes in the latest version of the **spatstat** package.

Usage

```
bugfixes(sinceversion = NULL, sincedate = NULL,
         package = "spatstat", show = TRUE)
```

Arguments

sinceversion	Earliest version of package for which bugs should be listed. The default is the current installed version.
sincedate	Earliest release date of package for which bugs should be listed. A character string or a date-time object.
package	Character string. The name of the package for which bugs are to be listed.
show	Logical value indicating whether to display the bug table on the terminal.

Details

Bug reports are extracted from the NEWS file of the specified package. Only those after a specified date, or after a specified version of the package, are retained. The bug reports are then sorted alphabetically, so that all bugs affecting a particular function are listed consecutively. Finally the table of bug reports is displayed (if show=TRUE) and returned invisibly.

The argument sinceversion should be a character string like "1.2-3". The default is the current installed version of the package. The argument sincedata should be a character string like "2015-05-27", or a date-time object.

Typing bugfixes without parentheses will display a table of all bug fixes in the current installed version of **spatstat**.

Value

A data frame, belonging to the class "bugtable", which has its own print method.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

latest.news, news.

Examples

```
# show all bugs reported after publication of the spatstat book
if(interactive()) bugfixes("1.42-0")
```

---

bw.diggle *Cross Validated Bandwidth Selection for Kernel Density*

Description

Uses cross-validation to select a smoothing bandwidth for the kernel estimation of point process intensity.

Usage

```
bw.diggle(X, ..., correction="good", hmax=NULL, nr=512)
```

Arguments

X	A point pattern (object of class "ppp").
...	Ignored.
correction	Character string passed to Kest determining the edge correction to be used to calculate the K function.
hmax	Numeric. Maximum value of bandwidth that should be considered.
nr	Integer. Number of steps in the distance value r to use in computing numerical integrals.

Details

This function selects an appropriate bandwidth sigma for the kernel estimator of point process intensity computed by density.ppp.

The bandwidth σ is chosen to minimise the mean-square error criterion defined by Diggle (1985). The algorithm uses the method of Berman and Diggle (1989) to compute the quantity

$$M(\sigma) = \frac{\mathrm{MSE}(\sigma)}{\lambda^2} - g(0)$$

as a function of bandwidth σ, where $\mathrm{MSE}(\sigma)$ is the mean squared error at bandwidth σ, while λ is the mean intensity, and g is the pair correlation function. See Diggle (2003, pages 115-118) for a summary of this method.

The result is a numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" which can be plotted to show the (rescaled) mean-square error as a function of sigma.

Value

A numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" which can be plotted.

Definition of bandwidth

The smoothing parameter `sigma` returned by `bw.diggle` (and displayed on the horizontal axis of the plot) corresponds to `h/2`, where `h` is the smoothing parameter described in Diggle (2003, pages 116-118) and Berman and Diggle (1989). In those references, the smoothing kernel is the uniform density on the disc of radius `h`. In `density.ppp`, the smoothing kernel is the isotropic Gaussian density with standard deviation `sigma`. When replacing one kernel by another, the usual practice is to adjust the bandwidths so that the kernels have equal variance (cf. Diggle 2003, page 118). This implies that `sigma = h/2`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Berman, M. and Diggle, P. (1989) Estimating weighted integrals of the second-order intensity of a spatial point process. *Journal of the Royal Statistical Society, series B* **51**, 81–92.

Diggle, P.J. (1985) A kernel method for smoothing point process data. *Applied Statistics* (Journal of the Royal Statistical Society, Series C) **34** (1985) 138–147.

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

See Also

`density.ppp`, `bw.ppl`, `bw.scott`

Examples

```
data(lansing)
attach(split(lansing))
b <- bw.diggle(hickory)
plot(b, ylim=c(-2, 0), main="Cross validation for hickories")

 plot(density(hickory, b))
```

`bw.frac` *Bandwidth Selection Based on Window Geometry*

Description

Select a smoothing bandwidth for smoothing a point pattern, based only on the geometry of the spatial window. The bandwidth is a specified quantile of the distance between two independent random points in the window.

Usage

```
bw.frac(X, ..., f=1/4)
```

Arguments

X	A window (object of class "owin") or point pattern (object of class "ppp") or other data which can be converted to a window using as.owin.
...	Arguments passed to distcdf.
f	Probability value (between 0 and 1) determining the quantile of the distribution.

Details

This function selects an appropriate bandwidth sigma for the kernel estimator of point process intensity computed by density.ppp.

The bandwidth σ is computed as a quantile of the distance between two independent random points in the window. The default is the lower quartile of this distribution.

If $F(r)$ is the cumulative distribution function of the distance between two independent random points uniformly distributed in the window, then the value returned is the quantile with probability f. That is, the bandwidth is the value r such that $F(r) = f$.

The cumulative distribution function $F(r)$ is computed using distcdf. We then we compute the smallest number r such that $F(r) \geq f$.

Value

A numerical value giving the selected bandwidth. The result also belongs to the class "bw.frac" which can be plotted to show the cumulative distribution function and the selected quantile.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

density.ppp, bw.diggle, bw.ppl, bw.relrisk, bw.scott, bw.smoothppp, bw.stoyan

Examples

```
h <- bw.frac(letterR)
h
plot(h, main="bw.frac(letterR)")
```

---

bw.pcf — *Cross Validated Bandwidth Selection for Pair Correlation Function*

Description

Uses composite likelihood or generalized least squares cross-validation to select a smoothing bandwidth for the kernel estimation of pair correlation function.

Usage

```
bw.pcf(X, rmax=NULL, lambda=NULL, divisor="r",
       kernel="epanechnikov", nr=10000, bias.correct=TRUE,
       cv.method=c("compLik", "leastSQ"), simple=TRUE, srange=NULL,
 ..., verbose=FALSE)
```

Arguments

X	A point pattern (object of class "ppp").
rmax	Numeric. Maximum value of the spatial lag distance r for which $g(r)$ should be evaluated.
lambda	Optional. Values of the estimated intensity function. A vector giving the intensity values at the points of the pattern X.
divisor	Choice of divisor in the estimation formula: either "r" (the default) or "d". See pcf.ppp.
kernel	Choice of smoothing kernel, passed to density; see pcf and pcfinhom.
nr	Integer. Number of subintervals for discretization of [0, rmax] to use in computing numerical integrals.
bias.correct	Logical. Whether to use bias corrected version of the kernel estimate. See Details.
cv.method	Choice of cross validation method: either "compLik" or "leastSQ" (partially matched).
simple	Logical. Whether to use simple removal of spatial lag distances. See Details.
srange	Optional. Numeric vector of length 2 giving the range of bandwidth values that should be searched to find the optimum bandwidth.
...	Other arguments, passed to pcf or pcfinhom.
verbose	Logical value indicating whether to print progress reports during the optimization procedure.

Details

This function selects an appropriate bandwidth bw for the kernel estimator of the pair correlation function of a point process intensity computed by pcf.ppp (homogeneous case) or pcfinhom (inhomogeneous case).

With cv.method="leastSQ", the bandwidth h is chosen to minimise an unbiased estimate of the integrated mean-square error criterion $M(h)$ defined in equation (4) in Guan (2007a).

With cv.method="compLik", the bandwidth h is chosen to maximise a likelihood cross-validation criterion $CV(h)$ defined in equation (6) of Guan (2007b).

$$M(b) = \frac{\text{MSE}(\sigma)}{\lambda^2} - g(0)$$

The result is a numerical value giving the selected bandwidth.

Value

A numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" which can be plotted.

Definition of bandwidth

The bandwidth `bw` returned by `bw.pcf` corresponds to the standard deviation of the smoothoing kernel. As mentioned in the documentation of `density.default` and `pcf.ppp`, this differs from the scale parameter `h` of the smoothing kernel which is often considered in the literature as the bandwidth of the kernel function. For example for the Epanechnikov kernel, `bw=h/sqrt(h)`.

Author(s)

Rasmus Waagepetersen and Abdollah Jalilian. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.ed
Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Guan, Y. (2007a). A composite likelihood cross-validation approach in selecting bandwidth for the estimation of the pair correlation function. *Scandinavian Journal of Statistics*, **34**(2), 336–346.

Guan, Y. (2007b). A least-squares cross-validation bandwidth selection approach in pair correlation function estimations. *Statistics & Probability Letters*, **77**(18), 1722–1729.

See Also

`pcf.ppp`, `pcfinhom`

Examples

```
b <- bw.pcf(redwood)
plot(pcf(redwood, bw=b))
```

`bw.ppl` *Likelihood Cross Validation Bandwidth Selection for Kernel Density*

Description

Uses likelihood cross-validation to select a smoothing bandwidth for the kernel estimation of point process intensity.

Usage

```
bw.ppl(X, ..., srange=NULL, ns=16, sigma=NULL, weights=NULL)
```

Arguments

`X`	A point pattern (object of class `"ppp"`).
`...`	Ignored.
`srange`	Optional numeric vector of length 2 giving the range of values of bandwidth to be searched.
`ns`	Optional integer giving the number of values of bandwidth to search.
`sigma`	Optional. Vector of values of the bandwidth to be searched. Overrides the values of `ns` and `srange`.
`weights`	Optional. Numeric vector of weights for the points of `X`. Argument passed to `density.ppp`.

Details

This function selects an appropriate bandwidth `sigma` for the kernel estimator of point process intensity computed by `density.ppp`.

The bandwidth σ is chosen to maximise the point process likelihood cross-validation criterion

$$\mathrm{LCV}(\sigma) = \sum_i \log \hat{\lambda}_{-i}(x_i) - \int_W \hat{\lambda}(u)\,\mathrm{d}u$$

where the sum is taken over all the data points x_i, where $\hat{\lambda}_{-i}(x_i)$ is the leave-one-out kernel-smoothing estimate of the intensity at x_i with smoothing bandwidth σ, and $\hat{\lambda}(u)$ is the kernel-smoothing estimate of the intensity at a spatial location u with smoothing bandwidth σ. See Loader(1999, Section 5.3).

The value of $\mathrm{LCV}(\sigma)$ is computed directly, using `density.ppp`, for `ns` different values of σ between `srange[1]` and `srange[2]`.

The result is a numerical value giving the selected bandwidth. The result also belongs to the class `"bw.optim"` which can be plotted to show the (rescaled) mean-square error as a function of `sigma`.

Value

A numerical value giving the selected bandwidth. The result also belongs to the class `"bw.optim"` which can be plotted.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Loader, C. (1999) *Local Regression and Likelihood.* Springer, New York.

See Also

`density.ppp`, `bw.diggle`, `bw.scott`

Examples

```
b <- bw.ppl(redwood)
plot(b, main="Likelihood cross validation for redwoods")
plot(density(redwood, b))
```

bw.relrisk *Cross Validated Bandwidth Selection for Relative Risk Estimation*

Description

Uses cross-validation to select a smoothing bandwidth for the estimation of relative risk.

Usage

```
bw.relrisk(X, method = "likelihood", nh = spatstat.options("n.bandwidth"),
hmin=NULL, hmax=NULL, warn=TRUE)
```

Arguments

`X`	A multitype point pattern (object of class "ppp" which has factor valued marks).
`method`	Character string determining the cross-validation method. Current options are "likelihood", "leastsquares" or "weightedleastsquares".
`nh`	Number of trial values of smoothing bandwith `sigma` to consider. The default is 32.
`hmin, hmax`	Optional. Numeric values. Range of trial values of smoothing bandwith `sigma` to consider. There is a sensible default.
`warn`	Logical. If `TRUE`, issue a warning if the minimum of the cross-validation criterion occurs at one of the ends of the search interval.

Details

This function selects an appropriate bandwidth for the nonparametric estimation of relative risk using `relrisk`.

Consider the indicators y_{ij} which equal 1 when data point x_i belongs to type j, and equal 0 otherwise. For a particular value of smoothing bandwidth, let $\hat{p}_j(u)$ be the estimated probabilities that a point at location u will belong to type j. Then the bandwidth is chosen to minimise either the likelihood, the squared error, or the approximately standardised squared error, of the indicators y_{ij} relative to the fitted values $\hat{p}_j(x_i)$. See Diggle (2003).

The result is a numerical value giving the selected bandwidth `sigma`. The result also belongs to the class "bw.optim" allowing it to be printed and plotted. The plot shows the cross-validation criterion as a function of bandwidth.

The range of values for the smoothing bandwidth `sigma` is set by the arguments `hmin, hmax`. There is a sensible default, based on multiples of Stoyan's rule of thumb `bw.stoyan`.

If the optimal bandwidth is achieved at an endpoint of the interval `[hmin, hmax]`, the algorithm will issue a warning (unless `warn=FALSE`). If this occurs, then it is probably advisable to expand the interval by changing the arguments `hmin, hmax`.

Computation time depends on the number `nh` of trial values considered, and also on the range `[hmin, hmax]` of values considered, because larger values of `sigma` require calculations involving more pairs of data points.

Value

A numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" which can be plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

References

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

Kelsall, J.E. and Diggle, P.J. (1995) Kernel estimation of relative risk. *Bernoulli* **1**, 3–16.

See Also

relrisk, bw.stoyan

Examples

```
data(urkiola)

b <- bw.relrisk(urkiola)
b
plot(b)
b <- bw.relrisk(urkiola, hmax=20)
plot(b)
```

---

bw.scott *Scott's Rule for Bandwidth Selection for Kernel Density*

Description

Use Scott's rule of thumb to determine the smoothing bandwidth for the kernel estimation of point process intensity.

Usage

```
bw.scott(X)
```

Arguments

X	A point pattern (object of class "ppp").

Details

This function selects a bandwidth sigma for the kernel estimator of point process intensity computed by density.ppp.

The bandwidth σ is computed by the rule of thumb of Scott (1992, page 152). It is very fast to compute.

This rule is designed for density estimation, and typically produces a larger bandwidth than bw.diggle. It is useful for estimating gradual trend.

Value

A numerical vector of two elements giving the selected bandwidths in the x and y directions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Scott, D.W. (1992) *Multivariate Density Estimation. Theory, Practice and Visualization*. New York: Wiley.

See Also

density.ppp, bw.diggle, bw.ppl, bw.frac.

Examples

```
data(lansing)
attach(split(lansing))
b <- bw.scott(hickory)
b

 plot(density(hickory, b))
```

---

bw.smoothppp *Cross Validated Bandwidth Selection for Spatial Smoothing*

Description

Uses least-squares cross-validation to select a smoothing bandwidth for spatial smoothing of marks.

Usage

```
bw.smoothppp(X, nh = spatstat.options("n.bandwidth"),
hmin=NULL, hmax=NULL, warn=TRUE)
```

Arguments

X	A marked point pattern with numeric marks.
nh	Number of trial values of smoothing bandwith `sigma` to consider. The default is 32.
hmin, hmax	Optional. Numeric values. Range of trial values of smoothing bandwith `sigma` to consider. There is a sensible default.
warn	Logical. If `TRUE`, issue a warning if the minimum of the cross-validation criterion occurs at one of the ends of the search interval.

Details

This function selects an appropriate bandwidth for the nonparametric smoothing of mark values using Smooth.ppp.

The argument X must be a marked point pattern with a vector or data frame of marks. All mark values must be numeric.

The bandwidth is selected by least-squares cross-validation. Let y_i be the mark value at the ith data point. For a particular choice of smoothing bandwidth, let $\hat{y}_i$ be the smoothed value at the ith data point. Then the bandwidth is chosen to minimise the squared error of the smoothed values $\sum_i (y_i - \hat{y}_i)^2$.

The result of bw.smoothppp is a numerical value giving the selected bandwidth sigma. The result also belongs to the class "bw.optim" allowing it to be printed and plotted. The plot shows the cross-validation criterion as a function of bandwidth.

The range of values for the smoothing bandwidth sigma is set by the arguments hmin, hmax. There is a sensible default, based on the nearest neighbour distances.

If the optimal bandwidth is achieved at an endpoint of the interval [hmin, hmax], the algorithm will issue a warning (unless warn=FALSE). If this occurs, then it is probably advisable to expand the interval by changing the arguments hmin, hmax.

Computation time depends on the number nh of trial values considered, and also on the range [hmin, hmax] of values considered, because larger values of sigma require calculations involving more pairs of data points.

Value

A numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" which can be plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Smooth.ppp

Examples

```
data(longleaf)

b <- bw.smoothppp(longleaf)
b
plot(b)
```

bw.stoyan	*Stoyan's Rule of Thumb for Bandwidth Selection*

Description

Computes a rough estimate of the appropriate bandwidth for kernel smoothing estimators of the pair correlation function and other quantities.

Usage

```
bw.stoyan(X, co=0.15)
```

Arguments

`X`	A point pattern (object of class "ppp").
`co`	Coefficient appearing in the rule of thumb. See Details.

Details

Estimation of the pair correlation function and other quantities by smoothing methods requires a choice of the smoothing bandwidth. Stoyan and Stoyan (1995, equation (15.16), page 285) proposed a rule of thumb for choosing the smoothing bandwidth.

For the Epanechnikov kernel, the rule of thumb is to set the kernel's half-width h to $0.15/\sqrt{\lambda}$ where λ is the estimated intensity of the point pattern, typically computed as the number of points of `X` divided by the area of the window containing `X`.

For a general kernel, the corresponding rule is to set the standard deviation of the kernel to $\sigma = 0.15/\sqrt{5\lambda}$.

The coefficient 0.15 can be tweaked using the argument `co`.

Value

A numerical value giving the selected bandwidth (the standard deviation of the smoothing kernel).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Stoyan, D. and Stoyan, H. (1995) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

pcf, bw.relrisk

Examples

```
data(shapley)
bw.stoyan(shapley)
```

by.im *Apply Function to Image Broken Down by Factor*

Description

Splits a pixel image into sub-images and applies a function to each sub-image.

Usage

```
## S3 method for class 'im'
by(data, INDICES, FUN, ...)
```

Arguments

`data`	A pixel image (object of class `"im"`).
`INDICES`	Grouping variable. Either a tessellation (object of class `"tess"`) or a factor-valued pixel image.
`FUN`	Function to be applied to each sub-image of `data`.
`...`	Extra arguments passed to `FUN`.

Details

This is a method for the generic function `by` for pixel images (class `"im"`).

The pixel image `data` is first divided into sub-images according to `INDICES`. Then the function `FUN` is applied to each subset. The results of each computation are returned in a list.

The grouping variable `INDICES` may be either

- a tessellation (object of class `"tess"`). Each tile of the tessellation delineates a subset of the spatial domain.
- a pixel image (object of class `"im"`) with factor values. The levels of the factor determine subsets of the spatial domain.

Value

A list containing the results of each evaluation of `FUN`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`split.im`, `tess`, `im`

Examples

```
W <- square(1)
X <- as.im(function(x,y){sqrt(x^2+y^2)}, W)
Y <- dirichlet(runifpoint(12, W))
# mean pixel value in each subset
unlist(by(X, Y, mean))
# trimmed mean
unlist(by(X, Y, mean, trim=0.05))
```

`by.ppp` *Apply a Function to a Point Pattern Broken Down by Factor*

Description

Splits a point pattern into sub-patterns, and applies the function to each sub-pattern.

Usage

```
## S3 method for class 'ppp'
by(data, INDICES=marks(data), FUN, ...)
```

Arguments

`data`	Point pattern (object of class "ppp").
`INDICES`	Grouping variable. Either a factor, a pixel image with factor values, or a tessellation.
`FUN`	Function to be applied to subsets of `data`.
`...`	Additional arguments to `FUN`.

Details

This is a method for the generic function `by` for point patterns (class "ppp").

The point pattern `data` is first divided into subsets according to `INDICES`. Then the function `FUN` is applied to each subset. The results of each computation are returned in a list.

The argument `INDICES` may be

- a factor, of length equal to the number of points in `data`. The levels of `INDICES` determine the destination of each point in `data`. The `i`th point of `data` will be placed in the sub-pattern `split.ppp(data)$l` where `l = f[i]`.
- a pixel image (object of class "im") with factor values. The pixel value of `INDICES` at each point of `data` will be used as the classifying variable.
- a tessellation (object of class "tess"). Each point of `data` will be classified according to the tile of the tessellation into which it falls.

If `INDICES` is missing, then `data` must be a multitype point pattern (a marked point pattern whose marks vector is a factor). Then the effect is that the points of each type are separated into different point patterns.

Value

A list (also of class "anylist" or "solist" as appropriate) containing the results returned from FUN for each of the subpatterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppp, split.ppp, cut.ppp, tess, im.

Examples

```
# multitype point pattern, broken down by type
data(amacrine)
by(amacrine, FUN=density)
by(amacrine, FUN=function(x) { min(nndist(x)) } )

# how to pass additional arguments to FUN
by(amacrine, FUN=clarkevans, correction=c("Donnelly","cdf"))

# point pattern broken down by tessellation
data(swedishpines)
tes <- quadrats(swedishpines, 5, 5)
B <- by(swedishpines, tes, clarkevans, correction="Donnelly")
unlist(lapply(B, as.numeric))
```

---

cauchy.estK *Fit the Neyman-Scott cluster process with Cauchy kernel*

Description

Fits the Neyman-Scott Cluster point process with Cauchy kernel to a point pattern dataset by the Method of Minimum Contrast.

Usage

```
cauchy.estK(X, startpar=c(kappa=1,scale=1), lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...)
```

Arguments

X	Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the model.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.

Details

This algorithm fits the Neyman-Scott cluster point process model with Cauchy kernel to a point pattern dataset by the Method of Minimum Contrast, using the K function.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The K function of the point pattern will be computed using Kest, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the K function, and this object should have been obtained by a call to Kest or one of its relatives.

The algorithm fits the Neyman-Scott cluster point process with Cauchy kernel to X, by finding the parameters of the Matern Cluster model which give the closest match between the theoretical K function of the Matern Cluster process and the observed K function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The model is described in Jalilian et al (2013). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent follow a common distribution described in Jalilian et al (2013).

If the argument lambda is provided, then this is used as the value of the point process intensity λ. Otherwise, if X is a point pattern, then λ will be estimated from X. If X is a summary statistic and lambda is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as NA.

The remaining arguments rmin,rmax,q,p control the method of minimum contrast; see mincontrast.

The corresponding model can be simulated using rCauchy.

For computational reasons, the optimisation procedure uses the parameter eta2, which is equivalent to 4 * scale^2 where scale is the scale parameter for the model as used in rCauchy.

Homogeneous or inhomogeneous Neyman-Scott/Cauchy models can also be fitted using the function kppm and the fitted models can be simulated using simulate.kppm.

The optimisation algorithm can be controlled through the additional arguments "..." which are passed to the optimisation function optim. For example, to constrain the parameter values to a certain range, use the argument method="L-BFGS-B" to select an optimisation algorithm that respects box constraints, and use the arguments lower and upper to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following main components:

par	Vector of fitted parameter values.
fit	Function value table (object of class "fv") containing the observed values of the summary statistic (observed) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.ed

References

Ghorbani, M. (2012) Cauchy cluster process. *Metrika*, to appear.

Jalilian, A., Guan, Y. and Waagepetersen, R. (2013) Decomposition of variance for spatial Cox processes. *Scandinavian Journal of Statistics* **40**, 119-137.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

kppm, cauchy.estpcf, lgcp.estK, thomas.estK, vargamma.estK, mincontrast, Kest, Kmodel.

rCauchy to simulate the model.

Examples

```
u <- cauchy.estK(redwood)
u
plot(u)
```

`cauchy.estpcf` *Fit the Neyman-Scott cluster process with Cauchy kernel*

Description

Fits the Neyman-Scott Cluster point process with Cauchy kernel to a point pattern dataset by the Method of Minimum Contrast, using the pair correlation function.

Usage

```
cauchy.estpcf(X, startpar=c(kappa=1,scale=1), lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...,
            pcfargs = list())
```

Arguments

`X`	Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
`startpar`	Vector of starting values for the parameters of the model.
`lambda`	Optional. An estimate of the intensity of the point process.
`q,p`	Optional. Exponents for the contrast criterion.
`rmin, rmax`	Optional. The interval of r values for the contrast criterion.
`...`	Optional arguments passed to optim to control the optimisation algorithm. See Details.
`pcfargs`	Optional list containing arguments passed to pcf.ppp to control the smoothing in the estimation of the pair correlation function.

Details

This algorithm fits the Neyman-Scott cluster point process model with Cauchy kernel to a point pattern dataset by the Method of Minimum Contrast, using the pair correlation function.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The pair correlation function of the point pattern will be computed using pcf, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the pair correlation function, and this object should have been obtained by a call to pcf or one of its relatives.

The algorithm fits the Neyman-Scott cluster point process with Cauchy kernel to X, by finding the parameters of the Matern Cluster model which give the closest match between the theoretical pair correlation function of the Matern Cluster process and the observed pair correlation function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The model is described in Jalilian et al (2013). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent follow a common distribution described in Jalilian et al (2013).

If the argument lambda is provided, then this is used as the value of the point process intensity λ. Otherwise, if X is a point pattern, then λ will be estimated from X. If X is a summary statistic and lambda is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as NA.

The remaining arguments rmin,rmax,q,p control the method of minimum contrast; see mincontrast.

The corresponding model can be simulated using rCauchy.

For computational reasons, the optimisation procedure internally uses the parameter eta2, which is equivalent to 4 * scale^2 where scale is the scale parameter for the model as used in rCauchy.

Homogeneous or inhomogeneous Neyman-Scott/Cauchy models can also be fitted using the function kppm and the fitted models can be simulated using simulate.kppm.

The optimisation algorithm can be controlled through the additional arguments "..." which are passed to the optimisation function optim. For example, to constrain the parameter values to a certain range, use the argument method="L-BFGS-B" to select an optimisation algorithm that respects box constraints, and use the arguments lower and upper to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following main components:

par	Vector of fitted parameter values.
fit	Function value table (object of class "fv") containing the observed values of the summary statistic (observed) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.ed

References

Ghorbani, M. (2012) Cauchy cluster process. *Metrika*, to appear.

Jalilian, A., Guan, Y. and Waagepetersen, R. (2013) Decomposition of variance for spatial Cox processes. *Scandinavian Journal of Statistics* **40**, 119-137.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

kppm, cauchy.estK, lgcp.estpcf, thomas.estpcf, vargamma.estpcf, mincontrast, pcf, pcfmodel.

rCauchy to simulate the model.

Examples

```
u <- cauchy.estpcf(redwood)
u
plot(u, legendpos="topright")
```

---

cbind.hyperframe   *Combine Hyperframes by Rows or by Columns*

Description

Methods for cbind and rbind for hyperframes.

Usage

```
## S3 method for class 'hyperframe'
cbind(...)
## S3 method for class 'hyperframe'
rbind(...)
```

Arguments

...	Any number of hyperframes (objects of class hyperframe).

Details

These are methods for cbind and rbind for hyperframes.

Note that *all* the arguments must be hyperframes (because of the peculiar dispatch rules of cbind and rbind).

To combine a hyperframe with a data frame, one should either convert the data frame to a hyperframe using as.hyperframe, or explicitly invoke the function cbind.hyperframe or rbind.hyperframe.

In other words: if h is a hyperframe and d is a data frame, the result of cbind(h,d) will be the same as cbind(as.data.frame(h), d), so that all hypercolumns of h will be deleted (and a warning will be issued). To combine h with d so that all columns of h are retained, type either cbind(h, as.hyperframe(d)) or cbind.hyperframe(h,d).

Value

Another hyperframe.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

hyperframe, as.hyperframe

Examples

```
lambda <- runif(5, min=10, max=30)
X <- lapply(as.list(lambda), function(x) { rpoispp(x) })
h <- hyperframe(lambda=lambda, X=X)
g <- hyperframe(id=letters[1:5], Y=rev(X))
gh <- cbind(h, g)
hh <- rbind(h, h)
```

CDF *Cumulative Distribution Function From Kernel Density Estimate*

Description

Given a kernel estimate of a probability density, compute the corresponding cumulative distribution function.

Usage

```
CDF(f, ...)

## S3 method for class 'density'
CDF(f, ..., warn = TRUE)
```

Arguments

f	Density estimate (object of class "density").
...	Ignored.
warn	Logical value indicating whether to issue a warning if the density estimate f had to be renormalised because it was computed in a restricted interval.

Details

CDF is generic, with a method for class "density".

This calculates the cumulative distribution function whose probability density has been estimated and stored in the object f. The object f must belong to the class "density", and would typically have been obtained from a call to the function density.

Value

A function, which can be applied to any numeric value or vector of values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

density, quantile.density

Examples

```
b <- density(runif(10))
f <- CDF(b)
f(0.5)
plot(f)
```

---

cdf.test *Spatial Distribution Test for Point Pattern or Point Process Model*

Description

Performs a test of goodness-of-fit of a point process model. The observed and predicted distributions of the values of a spatial covariate are compared using either the Kolmogorov-Smirnov test, Cramér-von Mises test or Anderson-Darling test. For non-Poisson models, a Monte Carlo test is used.

Usage

```
cdf.test(...)

## S3 method for class 'ppp'
cdf.test(X, covariate, test=c("ks", "cvm", "ad"), ...,
                       interpolate=TRUE, jitter=TRUE)

## S3 method for class 'ppm'
cdf.test(model, covariate,  test=c("ks", "cvm", "ad"), ...,
          interpolate=TRUE, jitter=TRUE, nsim=99, verbose=TRUE)

## S3 method for class 'lpp'
cdf.test(X, covariate,  test=c("ks", "cvm", "ad"), ...,
        interpolate=TRUE, jitter=TRUE)

## S3 method for class 'lppm'
cdf.test(model, covariate,  test=c("ks", "cvm", "ad"),
...,
          interpolate=TRUE, jitter=TRUE, nsim=99, verbose=TRUE)

## S3 method for class 'slrm'
cdf.test(model, covariate,  test=c("ks", "cvm", "ad"), ..., modelname=NULL, covname=NULL)
```

Arguments

X — A point pattern (object of class "ppp" or "lpp").

model — A fitted point process model (object of class "ppm" or "lppm") or fitted spatial logistic regression (object of class "slrm").

covariate — The spatial covariate on which the test will be based. A function, a pixel image (object of class "im"), a list of pixel images, or one of the characters "x" or "y" indicating the Cartesian coordinates.

test — Character string identifying the test to be performed: "ks" for Kolmogorov-Smirnov test, "cvm" for Cramér-von Mises test or "ad" for Anderson-Darling test.

... — Arguments passed to `ks.test` (from the **stats** package) or `cvm.test` or `ad.test` (from the **goftest** package) to control the test.

interpolate — Logical flag indicating whether to interpolate pixel images. If `interpolate=TRUE`, the value of the covariate at each point of X will be approximated by interpolating the nearby pixel values. If `interpolate=FALSE`, the nearest pixel value will be used.

jitter — Logical flag. If `jitter=TRUE`, values of the covariate will be slightly perturbed at random, to avoid tied values in the test.

modelname,covname — Character strings giving alternative names for `model` and `covariate` to be used in labelling plot axes.

nsim — Number of simulated realisations from the `model` to be used for the Monte Carlo test, when `model` is not a Poisson process.

verbose — Logical value indicating whether to print progress reports when performing a Monte Carlo test.

Details

These functions perform a goodness-of-fit test of a Poisson or Gibbs point process model fitted to point pattern data. The observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same values under the model, are compared using the Kolmogorov-Smirnov test, the Cramér-von Mises test or the Anderson-Darling test. For Gibbs models, a Monte Carlo test is performed using these test statistics.

The function `cdf.test` is generic, with methods for point patterns ("ppp" or "lpp"), point process models ("ppm" or "lppm") and spatial logistic regression models ("slrm").

- If X is a point pattern dataset (object of class "ppp"), then `cdf.test(X, ...)` performs a goodness-of-fit test of the uniform Poisson point process (Complete Spatial Randomness, CSR) for this dataset. For a multitype point pattern, the uniform intensity is assumed to depend on the type of point (sometimes called Complete Spatial Randomness and Independence, CSRI).
- If `model` is a fitted point process model (object of class "ppm" or "lppm") then `cdf.test(model, ...)` performs a test of goodness-of-fit for this fitted model.
- If `model` is a fitted spatial logistic regression (object of class "slrm") then `cdf.test(model, ...)` performs a test of goodness-of-fit for this fitted model.

The test is performed by comparing the observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same covariate under the model, using a classical goodness-of-fit test. Thus, you must nominate a spatial covariate for this test.

If X is a point pattern that does not have marks, the argument covariate should be either a function(x,y) or a pixel image (object of class "im" containing the values of a spatial function, or one of the characters "x" or "y" indicating the Cartesian coordinates. If covariate is an image, it should have numeric values, and its domain should cover the observation window of the model. If covariate is a function, it should expect two arguments x and y which are vectors of coordinates, and it should return a numeric vector of the same length as x and y.

If X is a multitype point pattern, the argument covariate can be either a function(x,y,marks), or a pixel image, or a list of pixel images corresponding to each possible mark value, or one of the characters "x" or "y" indicating the Cartesian coordinates.

First the original data point pattern is extracted from model. The values of the covariate at these data points are collected.

The predicted distribution of the values of the covariate under the fitted model is computed as follows. The values of the covariate at all locations in the observation window are evaluated, weighted according to the point process intensity of the fitted model, and compiled into a cumulative distribution function F using ewcdf.

The probability integral transformation is then applied: the values of the covariate at the original data points are transformed by the predicted cumulative distribution function F into numbers between 0 and 1. If the model is correct, these numbers are i.i.d. uniform random numbers. The A goodness-of-fit test of the uniform distribution is applied to these numbers using stats::ks.test, goftest::cvm.test or goftest::ad.test.

This test was apparently first described (in the context of spatial data, and using Kolmogorov-Smirnov) by Berman (1986). See also Baddeley et al (2005).

If model is not a Poisson process, then a Monte Carlo test is performed, by generating nsim point patterns which are simulated realisations of the model, re-fitting the model to each simulated point pattern, and calculating the test statistic for each fitted model. The Monte Carlo p value is determined by comparing the simulated values of the test statistic with the value for the original data.

The return value is an object of class "htest" containing the results of the hypothesis test. The print method for this class gives an informative summary of the test outcome.

The return value also belongs to the class "cdftest" for which there is a plot method plot.cdftest. The plot method displays the empirical cumulative distribution function of the covariate at the data points, and the predicted cumulative distribution function of the covariate under the model, plotted against the value of the covariate.

The argument jitter controls whether covariate values are randomly perturbed, in order to avoid ties. If the original data contains any ties in the covariate (i.e. points with equal values of the covariate), and if jitter=FALSE, then the Kolmogorov-Smirnov test implemented in ks.test will issue a warning that it cannot calculate the exact p-value. To avoid this, if jitter=TRUE each value of the covariate will be perturbed by adding a small random value. The perturbations are normally distributed with standard deviation equal to one hundredth of the range of values of the covariate. This prevents ties, and the p-value is still correct. There is a very slight loss of power.

Value

An object of class "htest" containing the results of the test. See ks.test for details. The return value can be printed to give an informative summary of the test.

The value also belongs to the class "cdftest" for which there is a plot method.

Warning

The outcome of the test involves a small amount of random variability, because (by default) the coordinates are randomly perturbed to avoid tied values. Hence, if cdf.test is executed twice, the

p-values will not be exactly the same. To avoid this behaviour, set `jitter=FALSE`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

Berman, M. (1986) Testing for spatial association between a point process and another stochastic process. *Applied Statistics* **35**, 54–62.

See Also

`plot.cdftest`, `quadrat.test`, `berman.test`, `ks.test`, `cvm.test`, `ad.test`, `ppm`

Examples

```
op <- options(useFancyQuotes=FALSE)

# test of CSR using x coordinate
cdf.test(nztrees, "x")
cdf.test(nztrees, "x", "cvm")
cdf.test(nztrees, "x", "ad")

# test of CSR using a function of x and y
fun <- function(x,y){2* x + y}
cdf.test(nztrees, fun)

# test of CSR using an image covariate
funimage <- as.im(fun, W=Window(nztrees))
cdf.test(nztrees, funimage)

# fit inhomogeneous Poisson model and test
model <- ppm(nztrees ~x)
cdf.test(model, "x")

if(interactive()) {
  # synthetic data: nonuniform Poisson process
  X <- rpoispp(function(x,y) { 100 * exp(x) }, win=square(1))

  # fit uniform Poisson process
  fit0 <- ppm(X ~1)
  # fit correct nonuniform Poisson process
  fit1 <- ppm(X ~x)

  # test wrong model
  cdf.test(fit0, "x")
  # test right model
  cdf.test(fit1, "x")
}

# multitype point pattern
cdf.test(amacrine, "x")
```

```
yimage <- as.im(function(x,y){y}, W=Window(amacrine))
cdf.test(ppm(amacrine ~marks+y), yimage)

options(op)
```

---

cdf.test.mppm *Spatial Distribution Test for Multiple Point Process Model*

Description

Performs a spatial distribution test of a point process model fitted to multiple spatial point patterns. The test compares the observed and predicted distributions of the values of a spatial covariate, using either the Kolmogorov-Smirnov, Cramér-von Mises or Anderson-Darling test of goodness-of-fit.

Usage

```
## S3 method for class 'mppm'
cdf.test(model, covariate, test=c("ks", "cvm", "ad"), ...,
            nsim=19, verbose=TRUE, interpolate=FALSE, fast=TRUE, jitter=TRUE)
```

Arguments

`model`	An object of class `"mppm"` representing a point process model fitted to multiple spatial point patterns.
`covariate`	The spatial covariate on which the test will be based. A function, a pixel image, a list of functions, a list of pixel images, a hyperframe, a character string containing the name of one of the covariates in `model`, or one of the strings `"x"` or `"y"`.
`test`	Character string identifying the test to be performed: `"ks"` for Kolmogorov-Smirnov test, `"cvm"` for Cramér-von Mises test or `"ad"` for Anderson-Darling test.
`...`	Arguments passed to `cdf.test` to control the test.
`nsim`	Number of simulated realisations which should be generated, if a Monte Carlo test is required.
`verbose`	Logical flag indicating whether to print progress reports.
`interpolate`	Logical flag indicating whether to interpolate between pixel values when code-covariate is a pixel image. See *Details*.
`fast`	Logical flag. If `TRUE`, values of the covariate are only sampled at the original quadrature points used to fit the model. If `FALSE`, values of the covariate are sampled at all pixels, which can be slower by three orders of magnitude.
`jitter`	Logical flag. If `TRUE`, observed values of the covariate are perturbed by adding small random values, to avoid tied observations.

Details

This function is a method for the generic function `cdf.test` for the class `mppm`.

This function performs a goodness-of-fit test of a point process model that has been fitted to multiple point patterns. The observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same values under the model, are compared using the Kolmogorov-Smirnov, Cramér-von Mises or Anderson-Darling test of goodness-of-fit. These are exact tests if the model is Poisson; otherwise, for a Gibbs model, a Monte Carlo p-value is computed by generating simulated realisations of the model and applying the selected goodness-of-fit test to each simulation.

The argument `model` should be a fitted point process model fitted to multiple point patterns (object of class `"mppm"`).

The argument `covariate` contains the values of a spatial function. It can be

- a `function(x,y)`
- a pixel image (object of class `"im"`
- a list of `function(x,y)`, one for each point pattern
- a list of pixel images, one for each point pattern
- a hyperframe (see `hyperframe`) of which the first column will be taken as containing the covariate
- a character string giving the name of one of the covariates in `model`
- one of the character strings `"x"` or `"y"`, indicating the spatial coordinates.

If `covariate` is an image, it should have numeric values, and its domain should cover the observation window of the `model`. If `covariate` is a function, it should expect two arguments `x` and `y` which are vectors of coordinates, and it should return a numeric vector of the same length as `x` and `y`.

First the original data point pattern is extracted from `model`. The values of the `covariate` at these data points are collected.

The predicted distribution of the values of the `covariate` under the fitted `model` is computed as follows. The values of the `covariate` at all locations in the observation window are evaluated, weighted according to the point process intensity of the fitted model, and compiled into a cumulative distribution function F using `ewcdf`.

The probability integral transformation is then applied: the values of the `covariate` at the original data points are transformed by the predicted cumulative distribution function F into numbers between 0 and 1. If the model is correct, these numbers are i.i.d. uniform random numbers. A goodness-of-fit test of the uniform distribution is applied to these numbers using `ks.test`, `cvm.test` or `ad.test`.

The argument `interpolate` determines how pixel values will be handled when codecovariate is a pixel image. The value of the covariate at a data point is obtained by looking up the value of the nearest pixel if `interpolate=FALSE`, or by linearly interpolating between the values of the four nearest pixels if `interpolate=TRUE`. Linear interpolation is slower, but is sometimes necessary to avoid tied values of the covariate arising when the pixel grid is coarse.

If `model` is a Poisson point process, then the Kolmogorov-Smirnov, Cramér-von Mises and Anderson-Darling tests are theoretically exact. This test was apparently first described (in the context of spatial data, and for Kolmogorov-Smirnov) by Berman (1986). See also Baddeley et al (2005).

If `model` is not a Poisson point process, then the Kolmogorov-Smirnov, Cramér-von Mises and Anderson-Darling tests are biased. Instead they are used as the basis of a Monte Carlo test. First `nsim` simulated realisations of the model will be generated. Each simulated realisation consists of a list of simulated point patterns, one for each of the original data patterns. This can take a very long time. The model is then re-fitted to each simulation, and the refitted model is subjected to the goodness-of-fit test described above. A Monte Carlo p-value is then computed by comparing the p-value of the original test with the p-values obtained from the simulations.

Value

An object of class "cdftest" and "htest" containing the results of the test. See cdf.test for details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

Baddeley, A., Turner, R., Moller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

Berman, M. (1986) Testing for spatial association between a point process and another stochastic process. *Applied Statistics* **35**, 54–62.

See Also

cdf.test, quadrat.test, mppm

Examples

```
# three i.i.d. realisations of nonuniform Poisson process
lambda <- as.im(function(x,y) { 300 * exp(x) }, square(1))
dat <- hyperframe(X=list(rpoispp(lambda), rpoispp(lambda), rpoispp(lambda)))

# fit uniform Poisson process
fit0 <- mppm(X~1, dat)
# fit correct nonuniform Poisson process
fit1 <- mppm(X~x, dat)

# test wrong model
cdf.test(fit0, "x")
# test right model
cdf.test(fit1, "x")
```

---

centroid.owin *Centroid of a window*

Description

Computes the centroid (centre of mass) of a window

Usage

```
centroid.owin(w, as.ppp = FALSE)
```

Arguments

w	A window
as.ppp	Logical flag indicating whether to return the centroid as a point pattern (ppp object)

Details

The centroid of the window w is computed. The centroid ("centre of mass") is the point whose x and y coordinates are the mean values of the x and y coordinates of all points in the window.

The argument w should be a window (an object of class "owin", see owin.object for details) or can be given in any format acceptable to as.owin().

The calculation uses an exact analytic formula for the case of polygonal windows.

Note that the centroid of a window is not necessarily inside the window, unless the window is convex. If as.ppp=TRUE and the centroid of w lies outside w, then the window of the returned point pattern will be a rectangle containing the original window (using as.rectangle.

Value

Either a list with components x, y, or a point pattern (of class ppp) consisting of a single point, giving the coordinates of the centroid of the window w.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

owin, as.owin

Examples

```
  w <- owin(c(0,1),c(0,1))
  centroid.owin(w)
  # returns 0.5, 0.5

  data(demopat)
  w <- Window(demopat)
  # an irregular window
  cent <- centroid.owin(w, as.ppp = TRUE)
  ## Not run:
  plot(cent)
  # plot the window and its centroid

## End(Not run)

  wapprox <- as.mask(w)
  # pixel approximation of window
```

```
## Not run:
points(centroid.owin(wapprox))
# should be indistinguishable

## End(Not run)
```

---

chop.tess *Subdivide a Window or Tessellation using a Set of Lines*

Description

Divide a given window into tiles delineated by a set of infinite straight lines, obtaining a tessellation of the window. Alternatively, given a tessellation, divide each tile of the tessellation into sub-tiles delineated by the lines.

Usage

```
chop.tess(X, L)
```

Arguments

X	A window (object of class "owin") or tessellation (object of class "tess") to be subdivided by lines.
L	A set of infinite straight lines (object of class "infline")

Details

The argument L should be a set of infinite straight lines in the plane (stored in an object L of class "infline" created by the function infline).

If X is a window, then it is divided into tiles delineated by the lines in L.

If X is a tessellation, then each tile of X is subdivided into sub-tiles delineated by the lines in L.

The result is a tessellation.

Value

A tessellation (object of class "tess").

Warning

If X is a non-convex window, or a tessellation containing non-convex tiles, then chop.tess(X,L) may contain a tile which consists of several unconnected pieces.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

infline, clip.infline

Examples

```
L <- infline(p=1:3, theta=pi/4)
W <- square(4)
chop.tess(W, L)
```

---

circdensity *Density Estimation for Circular Data*

Description

Computes a kernel smoothed estimate of the probability density for angular data.

Usage

```
circdensity(x, sigma = "nrd0", ...,
            bw = NULL,
            weights=NULL, unit = c("degree", "radian"))
```

Arguments

x	Numeric vector, containing angular data.
sigma	Smoothing bandwidth, or bandwidth selection rule, passed to density.default.
bw	Alternative to sigma for consistency with other functions.
...	Additional arguments passed to density.default, such as kernel and weights.
weights	Optional numeric vector of weights for the data in x.
unit	The unit of angle in which x is expressed.

Details

The angular values x are smoothed using (by default) the wrapped Gaussian kernel with standard deviation sigma.

Value

An object of class "density" (produced by density.default) which can be plotted by plot or by rose.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

density.default), rose.

Examples

```
ang <- runif(1000, max=360)
rose(circdensity(ang, 12))
```

`clarkevans` *Clark and Evans Aggregation Index*

Description

Computes the Clark and Evans aggregation index R for a spatial point pattern.

Usage

```
clarkevans(X, correction=c("none", "Donnelly", "cdf"),
              clipregion=NULL)
```

Arguments

`X`	A spatial point pattern (object of class "`ppp`").
`correction`	Character vector. The type of edge correction(s) to be applied.
`clipregion`	Clipping region for the guard area correction. A window (object of class "`owin`"). See Details.

Details

The Clark and Evans (1954) aggregation index R is a crude measure of clustering or ordering of a point pattern. It is the ratio of the observed mean nearest neighbour distance in the pattern to that expected for a Poisson point process of the same intensity. A value $R > 1$ suggests ordering, while $R < 1$ suggests clustering.

Without correction for edge effects, the value of `R` will be positively biased. Edge effects arise because, for a point of `X` close to the edge of the window, the true nearest neighbour may actually lie outside the window. Hence observed nearest neighbour distances tend to be larger than the true nearest neighbour distances.

The argument `correction` specifies an edge correction or several edge corrections to be applied. It is a character vector containing one or more of the options `"none"`, `"Donnelly"`, `"guard"` and `"cdf"` (which are recognised by partial matching). These edge corrections are:

"none": No edge correction is applied.

"Donnelly": Edge correction of Donnelly (1978), available for rectangular windows only. The theoretical expected value of mean nearest neighbour distance under a Poisson process is adjusted for edge effects by the edge correction of Donnelly (1978). The value of R is the ratio of the observed mean nearest neighbour distance to this adjusted theoretical mean.

"guard": Guard region or buffer area method. The observed mean nearest neighbour distance for the point pattern `X` is re-defined by averaging only over those points of `X` that fall inside the sub-window `clipregion`.

"cdf": Cumulative Distribution Function method. The nearest neighbour distance distribution function $G(r)$ of the stationary point process is estimated by `Gest` using the Kaplan-Meier type edge correction. Then the mean of the distribution is calculated from the cdf.

Alternatively `correction="all"` selects all options.

If the argument `clipregion` is given, then the selected edge corrections will be assumed to include `correction="guard"`.

To perform a test based on the Clark-Evans index, see `clarkevans.test`.

Value

A numeric value, or a numeric vector with named components

naive	R without edge correction
Donnelly	R using Donnelly edge correction
guard	R using guard region
cdf	R using cdf method

(as selected by correction). The value of the Donnelly component will be NA if the window of X is not a rectangle.

Author(s)

John Rudge <rudge@esc.cam.ac.uk> with modifications by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Clark, P.J. and Evans, F.C. (1954) Distance to nearest neighbour as a measure of spatial relationships in populations *Ecology* **35**, 445–453.

Donnelly, K. (1978) Simulations to determine the variance and edge-effect of total nearest neighbour distance. In I. Hodder (ed.) *Simulation studies in archaeology*, Cambridge/New York: Cambridge University Press, pp 91–95.

See Also

clarkevans.test, hopskel, nndist, Gest

Examples

```
# Example of a clustered pattern
clarkevans(redwood)

# Example of an ordered pattern
clarkevans(cells)

# Random pattern
X <- rpoispp(100)
clarkevans(X)

# How to specify a clipping region
clip1 <- owin(c(0.1,0.9),c(0.1,0.9))
clip2 <- erosion(Window(cells), 0.1)
clarkevans(cells, clipregion=clip1)
clarkevans(cells, clipregion=clip2)
```

clarkevans.test *Clark and Evans Test*

Description

Performs the Clark-Evans test of aggregation for a spatial point pattern.

Usage

```
clarkevans.test(X, ...,
                correction="none",
                clipregion=NULL,
                alternative=c("two.sided", "less", "greater",
                              "clustered", "regular"),
                nsim=999)
```

Arguments

`X`	A spatial point pattern (object of class "ppp").
`...`	Ignored.
`correction`	Character string. The type of edge correction to be applied. See `clarkevans`
`clipregion`	Clipping region for the guard area correction. A window (object of class "owin"). See `clarkevans`
`alternative`	String indicating the type of alternative for the hypothesis test. Partially matched.
`nsim`	Number of Monte Carlo simulations to perform, if a Monte Carlo p-value is required.

Details

This command uses the Clark and Evans (1954) aggregation index R as the basis for a crude test of clustering or ordering of a point pattern.

The Clark-Evans index is computed by the function `clarkevans`. See the help for `clarkevans` for information about the Clark-Evans index R and about the arguments `correction` and `clipregion`.

This command performs a hypothesis test of clustering or ordering of the point pattern `X`. The null hypothesis is Complete Spatial Randomness, i.e.\ a uniform Poisson process. The alternative hypothesis is specified by the argument `alternative`:

- `alternative="less"` or `alternative="clustered"`: the alternative hypothesis is that $R < 1$ corresponding to a clustered point pattern;
- `alternative="greater"` or `alternative="regular"`: the alternative hypothesis is that $R > 1$ corresponding to a regular or ordered point pattern;
- `alternative="two.sided"`: the alternative hypothesis is that $R \neq 1$ corresponding to a clustered or regular pattern.

The Clark-Evans index R is computed for the data as described in `clarkevans`.

If `correction="none"` and `nsim` is missing, the p-value for the test is computed by standardising R as proposed by Clark and Evans (1954) and referring the statistic to the standard Normal distribution.

Otherwise, the p-value for the test is computed by Monte Carlo simulation of `nsim` realisations of Complete Spatial Randomness conditional on the observed number of points.

Value

An object of class "htest" representing the result of the test.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Clark, P.J. and Evans, F.C. (1954) Distance to nearest neighbour as a measure of spatial relationships in populations. *Ecology* **35**, 445–453.

Donnelly, K. (1978) Simulations to determine the variance and edge-effect of total nearest neighbour distance. In *Simulation methods in archaeology*, Cambridge University Press, pp 91–95.

See Also

clarkevans, hopskel.test

Examples

```
# Redwood data - clustered
clarkevans.test(redwood)
clarkevans.test(redwood, alternative="clustered")
```

---

clickbox   *Interactively Define a Rectangle*

Description

Allows the user to specify a rectangle by point-and-click in the display.

Usage

```
clickbox(add=TRUE, ...)
```

Arguments

add	Logical value indicating whether to create a new plot (add=FALSE) or draw over the existing plot (add=TRUE).
...	Graphics arguments passed to polygon to plot the box.

Details

This function allows the user to create a rectangular window by interactively clicking on the screen display.

The user is prompted to point the mouse at any desired locations for two corners of the rectangle, and click the left mouse button to add each point.

The return value is a window (object of class "owin") representing the rectangle.

This function uses the R command locator to input the mouse clicks. It only works on screen devices such as 'X11', 'windows' and 'quartz'.

Value

A window (object of class "owin") representing the selected rectangle.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

clickpoly, clickppp, clickdist, locator

---

clickdist *Interactively Measure Distance*

Description

Measures the distance between two points which the user has clicked on.

Usage

```
clickdist()
```

Details

This function allows the user to measure the distance between two spatial locations, interactively, by clicking on the screen display.

When clickdist() is called, the user is expected to click two points in the current graphics device. The distance between these points will be returned.

This function uses the R command locator to input the mouse clicks. It only works on screen devices such as 'X11', 'windows' and 'quartz'.

Value

A single nonnegative number.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

locator, clickppp, clicklpp, clickpoly, clickbox `clickjoin` *Interactively join vertices on a plot*

Description

Given a point pattern representing a set of vertices, this command gives a point-and-click interface allowing the user to join pairs of selected vertices by edges.

Usage

```
clickjoin(X, ..., add = TRUE, m = NULL, join = TRUE)
```

Arguments

`X`	Point pattern of vertices. An object of class `"ppp"`.
`...`	Arguments passed to `segments` to control the plotting of the new edges.
`add`	Logical. Whether the point pattern `X` should be added to the existing plot (`add=TRUE`) or a new plot should be created (`add=FALSE`).
`m`	Optional. Logical matrix specifying an initial set of edges. There is an edge between vertices `i` and `j` if `m[i,j] = TRUE`.
`join`	Optional. If `TRUE`, then each user click will join a pair of vertices. If `FALSE`, then each user click will delete an existing edge. This is only relevant if `m` is supplied.

Details

This function makes it easier for the user to create a linear network or a planar graph, given a set of vertices.

The function first displays the point pattern `X`, then repeatedly prompts the user to click on a pair of points in `X`. Each selected pair of points will be joined by an edge. The function returns a logical matrix which has entries equal to `TRUE` for each pair of vertices joined by an edge.

The selection of points is performed using `identify.ppp` which typically expects the user to click the left mouse button. This point-and-click interaction continues until the user terminates it, by pressing the middle mouse button, or pressing the right mouse button and selecting `stop`.

The return value can be used in `linnet` to create a linear network.

Value

Logical matrix `m` with value `m[i,j] = TRUE` for every pair of vertices `X[i]` and `X[j]` that should be joined by an edge.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

See Also

`linnet`, `clickppp`

clicklpp *Interactively Add Points on a Linear Network*

Description

Allows the user to create a point pattern on a linear network by point-and-click in the display.

Usage

```
clicklpp(L, n=NULL, types=NULL, ...,
         add=FALSE, main=NULL, hook=NULL)
```

Arguments

L	Linear network on which the points will be placed. An object of class "linnet".
n	Number of points to be added (if this is predetermined).
types	Vector of types, when creating a multitype point pattern.
...	Optional extra arguments to be passed to locator to control the display.
add	Logical value indicating whether to create a new plot (add=FALSE) or draw over the existing plot (add=TRUE).
main	Main heading for plot.
hook	For internal use only. Do not use this argument.

Details

This function allows the user to create a point pattern on a linear network by interactively clicking on the screen display.

First the linear network L is plotted on the current screen device. Then the user is prompted to point the mouse at any desired locations and click the left mouse button to add each point. Interactive input stops after n clicks (if n was given) or when the middle mouse button is pressed.

The return value is a point pattern on the network L, containing the locations of all the clicked points, after they have been projected onto the network L. Any points that were clicked outside the bounding window of the network will be ignored.

If the argument types is given, then a multitype point pattern will be created. The user is prompted to input the locations of points of type type[i], for each successive index i. (If the argument n was given, there will be n points of *each* type.) The return value is a multitype point pattern on a linear network.

This function uses the R command locator to input the mouse clicks. It only works on screen devices such as 'X11', 'windows' and 'quartz'. Arguments that can be passed to locator through ... include pch (plotting character), cex (character expansion factor) and col (colour). See locator and par.

Value

A point pattern (object of class "lpp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>, based on an idea by Dominic Schuhmacher.

See Also

`clickppp`, `identify.lpp`, `locator`, `clickpoly`, `clickbox`, `clickdist`

---

clickpoly *Interactively Define a Polygon*

Description

Allows the user to create a polygon by point-and-click in the display.

Usage

```
clickpoly(add=FALSE, nv=NULL, np=1, ...)
```

Arguments

`add`	Logical value indicating whether to create a new plot (`add=FALSE`) or draw over the existing plot (`add=TRUE`).
`nv`	Number of vertices of the polygon (if this is predetermined).
`np`	Number of polygons to create.
`...`	Arguments passed to `locator` to control the interactive plot, and to `polygon` to plot the polygons.

Details

This function allows the user to create a polygonal window by interactively clicking on the screen display.

The user is prompted to point the mouse at any desired locations for the polygon vertices, and click the left mouse button to add each point. Interactive input stops after `nv` clicks (if `nv` was given) or when the middle mouse button is pressed.

The return value is a window (object of class `"owin"`) representing the polygon.

This function uses the R command `locator` to input the mouse clicks. It only works on screen devices such as 'X11', 'windows' and 'quartz'. Arguments that can be passed to `locator` through `...` include `pch` (plotting character), `cex` (character expansion factor) and `col` (colour). See `locator` and `par`.

Multiple polygons can also be drawn, by specifying `np > 1`. The polygons must be disjoint. The result is a single window object consisting of all the polygons.

Value

A window (object of class `"owin"`) representing the polygon.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

See Also

`identify.ppp`, `clickbox`, `clickppp`, `clickdist`, `locator`

`clickppp` *Interactively Add Points*

Description

Allows the user to create a point pattern by point-and-click in the display.

Usage

```
clickppp(n=NULL, win=square(1), types=NULL, ..., add=FALSE,
  main=NULL, hook=NULL)
```

Arguments

`n`	Number of points to be added (if this is predetermined).
`win`	Window in which to create the point pattern. An object of class `"owin"`.
`types`	Vector of types, when creating a multitype point pattern.
`...`	Optional extra arguments to be passed to `locator` to control the display.
`add`	Logical value indicating whether to create a new plot (`add=FALSE`) or draw over the existing plot (`add=TRUE`).
`main`	Main heading for plot.
`hook`	For internal use only. Do not use this argument.

Details

This function allows the user to create a point pattern by interactively clicking on the screen display.

First the window `win` is plotted on the current screen device. Then the user is prompted to point the mouse at any desired locations and click the left mouse button to add each point. Interactive input stops after `n` clicks (if `n` was given) or when the middle mouse button is pressed.

The return value is a point pattern containing the locations of all the clicked points inside the original window `win`, provided that all of the clicked locations were inside this window. Otherwise, the window is expanded to a box large enough to contain all the points (as well as containing the original window).

If the argument `types` is given, then a multitype point pattern will be created. The user is prompted to input the locations of points of type `type[i]`, for each successive index `i`. (If the argument `n` was given, there will be `n` points of *each* type.) The return value is a multitype point pattern.

This function uses the R command `locator` to input the mouse clicks. It only works on screen devices such as 'X11', 'windows' and 'quartz'. Arguments that can be passed to `locator` through `...` include `pch` (plotting character), `cex` (character expansion factor) and `col` (colour). See `locator` and `par`.

Value

A point pattern (object of class `"ppp"`).

Author(s)

Original by Dominic Schuhmacher. Adapted by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

See Also

identify.ppp, locator, clickpoly, clickbox, clickdist

---

clip.infline *Intersect Infinite Straight Lines with a Window*

Description

Take the intersection between a set of infinite straight lines and a window, yielding a set of line segments.

Usage

```
clip.infline(L, win)
```

Arguments

L	Object of class "infline" specifying a set of infinite straight lines in the plane.
win	Window (object of class "owin").

Details

This function computes the intersection between a set of infinite straight lines in the plane (stored in an object L of class "infline" created by the function infline) and a window win. The result is a pattern of line segments. Each line segment carries a mark indicating which line it belongs to.

Value

A line segment pattern (object of class "psp") with a single column of marks.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

See Also

infline,psp.

To divide a window into pieces using infinite lines, use chop.tess.

Examples

```
L <- infline(p=1:3, theta=pi/4)
W <- square(4)
clip.infline(L, W)
```

closepairs *Close Pairs of Points*

Description

Low-level functions to find all close pairs of points.

Usage

```
closepaircounts(X, r)

crosspaircounts(X, Y, r)

closepairs(X, rmax, ...)

## S3 method for class 'ppp'
closepairs(X, rmax, twice=TRUE,
                         what=c("all","indices","ijd"),
                         distinct=TRUE, neat=TRUE, ...)

crosspairs(X, Y, rmax, ...)

## S3 method for class 'ppp'
crosspairs(X, Y, rmax, what=c("all", "indices", "ijd"), ...)
```

Arguments

X,Y	Point patterns (objects of class "ppp").
r,rmax	Maximum distance between pairs of points to be counted as close pairs.
twice	Logical value indicating whether all ordered pairs of close points should be returned. If twice=TRUE (the default), each pair will appear twice in the output, as (i,j) and again as (j,i). If twice=FALSE, then each pair will appear only once, as the pair (i,j) with i < j.
what	String specifying the data to be returned for each close pair of points. If what="all" (the default) then the returned information includes the indices i,j of each pair, their x,y coordinates, and the distance between them. If what="indices" then only the indices i,j are returned. If what="ijd" then the indices i,j and the distance d are returned.
distinct	Logical value indicating whether to return only the pairs of points with different indices i and j (distinct=TRUE, the default) or to also include the pairs where i=j (distinct=FALSE).
neat	Logical value indicating whether to ensure that i < j in each output pair, when twice=FALSE.
...	Extra arguments, ignored by methods.

Details

These are the efficient low-level functions used by **spatstat** to find all close pairs of points in a point pattern or all close pairs between two point patterns.

closepaircounts(X,r) counts the number of neighbours for each point in the pattern X. That is, for each point X[i], it counts the number of other points X[j] with j != i such that d(X[i],X[j]) <= r where d denotes Euclidean distance. The result is an integer vector v such that v[i] is the number of neighbours of X[i].

crosspaircounts(X,Y,r) counts, for each point in the pattern X, the number of neighbours in the pattern Y. That is, for each point X[i], it counts the number of points Y[j] such that d(X[i],X[j]) <= r. The result is an integer vector v such that v[i] is the number of neighbours of X[i] in the pattern Y.

closepairs(X,rmax) identifies all pairs of distinct neighbours in the pattern X and returns them. The result is a list with the following components:

i Integer vector of indices of the first point in each pair.

j Integer vector of indices of the second point in each pair.

xi,yi Coordinates of the first point in each pair.

xj,yj Coordinates of the second point in each pair.

dx Equal to xj-xi

dy Equal to yj-yi

d Euclidean distance between each pair of points.

If what="indices" then only the components i and j are returned. This is slightly faster and more efficient with use of memory.

crosspairs(X,rmax) identifies all pairs of neighbours (X[i], Y[j]) between the patterns X and Y, and returns them. The result is a list with the same format as for closepairs.

Value

For closepaircounts and crosspaircounts, an integer vector of length equal to the number of points in X.

For closepairs and crosspairs, a list with components i and j, and possibly other components as described under Details.

Warning about accuracy

The results of these functions may not agree exactly with the correct answer (as calculated by a human) and may not be consistent between different computers and different installations of R. The discrepancies arise in marginal cases where the interpoint distance is equal to, or very close to, the threshold rmax.

Floating-point numbers in a computer are not mathematical Real Numbers: they are approximations using finite-precision binary arithmetic. The approximation is accurate to a tolerance of about .Machine$double.eps.

If the true interpoint distance d and the threshold rmax are equal, or if their difference is no more than .Machine$double.eps, the result may be incorrect.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

closepairs.pp3 for the corresponding functions for 3D point patterns.

Kest, Kcross, nndist, nncross, applynbd, markstat for functions which use these capabilities.

Examples

```
a <- closepaircounts(cells, 0.1)
sum(a)

Y <- split(amacrine)
b <- crosspaircounts(Y$on, Y$off, 0.1)

d <- closepairs(cells, 0.1)
e <- crosspairs(Y$on, Y$off, 0.1)
```

---

closepairs.pp3 *Close Pairs of Points in 3 Dimensions*

Description

Low-level functions to find all close pairs of points in three-dimensional point patterns.

Usage

```
## S3 method for class 'pp3'
closepairs(X, rmax, twice=TRUE,
                          what=c("all", "indices"),
                          distinct=TRUE, neat=TRUE, ...)

## S3 method for class 'pp3'
crosspairs(X, Y, rmax, what=c("all", "indices"), ...)
```

Arguments

X,Y	Point patterns in three dimensions (objects of class "pp3").
rmax	Maximum distance between pairs of points to be counted as close pairs.
twice	Logical value indicating whether all ordered pairs of close points should be returned. If twice=TRUE, each pair will appear twice in the output, as (i,j) and again as (j,i). If twice=FALSE, then each pair will appear only once, as the pair (i,j) such that i < j.
what	String specifying the data to be returned for each close pair of points. If what="all" (the default) then the returned information includes the indices i,j of each pair, their x,y,z coordinates, and the distance between them. If what="indices" then only the indices i,j are returned.
distinct	Logical value indicating whether to return only the pairs of points with different indices i and j (distinct=TRUE, the default) or to also include the pairs where i=j (distinct=FALSE).
neat	Logical value indicating whether to ensure that i < j in each output pair, when twice=FALSE.
...	Ignored.

Details

These are the efficient low-level functions used by **spatstat** to find all close pairs of points in a three-dimensional point pattern or all close pairs between two point patterns in three dimensions.

`closepairs(X,rmax)` identifies all pairs of neighbours in the pattern `X` and returns them. The result is a list with the following components:

i Integer vector of indices of the first point in each pair.

j Integer vector of indices of the second point in each pair.

xi,yi,zi Coordinates of the first point in each pair.

xj,yj,zj Coordinates of the second point in each pair.

dx Equal to `xj-xi`

dy Equal to `yj-yi`

dz Equal to `zj-zi`

d Euclidean distance between each pair of points.

If `what="indices"` then only the components `i` and `j` are returned. This is slightly faster.

`crosspairs(X,rmax)` identifies all pairs of neighbours `(X[i], Y[j])` between the patterns `X` and `Y`, and returns them. The result is a list with the same format as for `closepairs`.

Value

A list with components `i` and `j`, and possibly other components as described under Details.

Warning about accuracy

The results of these functions may not agree exactly with the correct answer (as calculated by a human) and may not be consistent between different computers and different installations of R. The discrepancies arise in marginal cases where the interpoint distance is equal to, or very close to, the threshold `rmax`.

Floating-point numbers in a computer are not mathematical Real Numbers: they are approximations using finite-precision binary arithmetic. The approximation is accurate to a tolerance of about `.Machine$double.eps`.

If the true interpoint distance d and the threshold `rmax` are equal, or if their difference is no more than `.Machine$double.eps`, the result may be incorrect.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

, Rolf Turner `<r.turner@auckland.ac.nz>`

and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`closepairs`

Examples

```
X <- pp3(runif(10), runif(10), runif(10), box3(c(0,1)))
Y <- pp3(runif(10), runif(10), runif(10), box3(c(0,1)))
a <- closepairs(X, 0.1)
b <- crosspairs(X, Y, 0.1)
```

closetriples *Close Triples of Points*

Description

Low-level function to find all close triples of points.

Usage

```
closetriples(X, rmax)
```

Arguments

X	Point pattern (object of class "ppp" or "pp3").
rmax	Maximum distance between each pair of points in a triple.

Details

This low-level function finds all triples of points in a point pattern in which each pair lies closer than rmax.

Value

A data frame with columns i,j,k giving the indices of the points in each triple, and a column diam giving the diameter (maximum pairwise distance) in the triple.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

closepairs, Tstat.

Examples

```
closetriples(redwoodfull, 0.02)
closetriples(redwoodfull, 0.005)
```

closing *Morphological Closing*

Description

Perform morphological closing of a window, a line segment pattern or a point pattern.

Usage

```
closing(w, r, ...)

## S3 method for class 'owin'
closing(w, r, ..., polygonal=NULL)

## S3 method for class 'ppp'
closing(w, r, ..., polygonal=TRUE)

## S3 method for class 'psp'
closing(w, r, ..., polygonal=TRUE)
```

Arguments

`w`	A window (object of class "owin" or a line segment pattern (object of class "psp") or a point pattern (object of class "ppp").
`r`	positive number: the radius of the closing.
`...`	extra arguments passed to `as.mask` controlling the pixel resolution, if a pixel approximation is used
`polygonal`	Logical flag indicating whether to compute a polygonal approximation to the erosion (`polygonal=TRUE`) or a pixel grid approximation (`polygonal=FALSE`).

Details

The morphological closing (Serra, 1982) of a set W by a distance $r > 0$ is the set of all points that cannot be separated from W by any circle of radius r. That is, a point x belongs to the closing $W*$ if it is impossible to draw any circle of radius r that has x on the inside and W on the outside. The closing $W*$ contains the original set W.

For a small radius r, the closing operation has the effect of smoothing out irregularities in the boundary of W. For larger radii, the closing operation smooths out concave features in the boundary. For very large radii, the closed set $W*$ becomes more and more convex.

The algorithm applies `dilation` followed by `erosion`.

Value

If `r > 0`, an object of class "owin" representing the closed region. If `r=0`, the result is identical to `w`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Serra, J. (1982) Image analysis and mathematical morphology. Academic Press.

See Also

opening for the opposite operation.

dilation, erosion for the basic operations.

owin, as.owin for information about windows.

Examples

```
v <- closing(letterR, 0.25)
plot(v, main="closing")
plot(letterR, add=TRUE)
```

---

clusterfield *Field of clusters*

Description

Calculate the superposition of cluster kernels at the location of a point pattern.

Usage

```
clusterfield(model, locations = NULL, ...)

## S3 method for class 'character'
clusterfield(model, locations = NULL, ...)

## S3 method for class 'function'
clusterfield(model, locations = NULL, ..., mu = NULL)

## S3 method for class 'kppm'
clusterfield(model, locations = NULL, ...)
```

Arguments

model	Cluster model. Either a fitted cluster model (object of class "kppm"), a character string specifying the type of cluster model, or a function defining the cluster kernel. See Details.
locations	A point pattern giving the locations of the kernels. Defaults to the centroid of the observation window for the "kppm" method and to the center of a unit square otherwise.
...	Additional arguments passed to density.ppp or the cluster kernel. See Details.
mu	Mean number of offspring per cluster. A single number or a pixel image.

Details

The actual calculations are preformed by density.ppp and ... arguments are passed thereto for control over the pixel resolution etc. (These arguments are then passed on to pixellate.ppp and as.mask.)

For the function method the given kernel function should accept vectors of x and y coordinates as its first two arguments. Any additional arguments may be passed through the ....

The function method also accepts the optional parameter mu (defaulting to 1) specifying the mean number of points per cluster (as a numeric) or the inhomogeneous reference cluster intensity (as an "im" object or a function(x,y)). The interpretation of mu is as explained in the simulation functions referenced in the See Also section below.

For the character method model must be one of: model="Thomas" for the Thomas process, model="MatClust" for the Matern cluster process, model="Cauchy" for the Neyman-Scott cluster process with Cauchy kernel, or model="VarGamma" for the Neyman-Scott cluster process with Variance Gamma kernel. For all these models the parameter scale is required and passed through ... as well as the parameter nu when model="VarGamma". This method calls clusterfield.function so the parameter mu may also be passed through ... and will be interpreted as explained above.

The kppm method extracts the relevant information from the fitted model (including mu) and calls clusterfield.function.

Value

A pixel image (object of class "im").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk> .

See Also

density.ppp and kppm

Simulation algorithms for cluster models: rCauchy rMatClust rThomas rVarGamma

Examples

```
# method for fitted model
fit <- kppm(redwood~1, "Thomas")
clusterfield(fit, eps = 0.01)

# method for functions
kernel <- function(x,y,scal) {
    r <- sqrt(x^2 + y^2)
    ifelse(r > 0,
           dgamma(r, shape=5, scale=scal)/(2 * pi * r),
           0)
}
X <- runifpoint(10)
clusterfield(kernel, X, scal=0.05)
```

`clusterfit` *Fit Cluster or Cox Point Process Model via Minimum Contrast*

Description

Fit a homogeneous or inhomogeneous cluster process or Cox point process model to a point pattern by the Method of Minimum Contrast.

Usage

```
clusterfit(X, clusters, lambda = NULL, startpar = NULL,
           q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...,
           statistic = NULL, statargs = NULL, algorithm="Nelder-Mead")
```

Arguments

`X`	Data to which the cluster or Cox model will be fitted. Either a point pattern or a summary statistic. See Details.
`clusters`	Character string determining the cluster or Cox model. Partially matched. Options are `"Thomas"`, `"MatClust"`, `"Cauchy"`, `"VarGamma"` and `"LGCP"`.
`lambda`	Optional. An estimate of the intensity of the point process. Either a single numeric specifying a constant intensity, a pixel image (object of class `"im"`) giving the intensity values at all locations, a fitted point process model (object of class `"ppm"` or `"kppm"`) or a `function(x,y)` which can be evaluated to give the intensity value at any location.
`startpar`	Vector of initial values of the parameters of the point process mode. If `X` is a point pattern sensible defaults are used. Otherwise rather arbitrary values are used.
`q,p`	Optional. Exponents for the contrast criterion.
`rmin, rmax`	Optional. The interval of r values for the contrast criterion.
`...`	Additional arguments passed to `mincontrast`.
`statistic`	Optional. Name of the summary statistic to be used for minimum contrast estimation: either `"K"` or `"pcf"`.
`statargs`	Optional list of arguments to be used when calculating the `statistic`. See Details.
`algorithm`	Character string determining the mathematical optimisation algorithm to be used by `optim`. See the argument `method` of `optim`.

Details

This function fits the clustering parameters of a cluster or Cox point process model by the Method of Minimum Contrast, that is, by matching the theoretical K-function of the model to the empirical K-function of the data, as explained in `mincontrast`.

If `statistic="pcf"` (or `X` appears to be an estimated pair correlation function) then instead of using the K-function, the algorithm will use the pair correlation function.

If `X` is a point pattern of class `"ppp"` an estimate of the summary statistic specfied by `statistic` (defaults to `"K"`) is first computed before minimum contrast estimation is carried out as described above. In this case the argument `statargs` can be used for controlling the summary statistic estimation. The precise algorithm for computing the summary statistic depends on whether the intensity specification (`lambda`) is:

homogeneous: If `lambda` is `NUll` or a single numeric the pattern is considered homogeneous and either `Kest` or `pcf` is invoked. In this case `lambda` is **not** used for anything when estimating the summary statistic.

inhomogeneous: If `lambda` is a pixel image (object of class `"im"`), a fitted point process model (object of class `"ppm"` or `"kppm"`) or a `function(x,y)` the pattern is considered inhomogeneous. In this case either `Kinhom` or `pcfinhom` is invoked with `lambda` as an argument.

After the clustering parameters of the model have been estimated by minimum contrast `lambda` (if non-null) is used to compute the additional model parameter μ.

Value

An object of class `"minconfit"`. There are methods for printing and plotting this object. See `mincontrast`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Waagepetersen, R. (2007). An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63** (2007) 252–258.

See Also

`kppm`

Examples

```
fit <- clusterfit(redwood, "Thomas")
fit
if(interactive()){
  plot(fit)
}
```

`clusterkernel` *Extract Cluster Offspring Kernel*

Description

Given a cluster point process model, this command returns the probability density of the cluster offspring.

Usage

```
clusterkernel(model, ...)

## S3 method for class 'kppm'
clusterkernel(model, ...)

## S3 method for class 'character'
clusterkernel(model, ...)
```

Arguments

`model`	Cluster model. Either a fitted cluster or Cox model (object of class `"kppm"`), or a character string specifying the type of cluster model.
`...`	Parameter values for the model, when `model` is a character string.

Details

Given a specification of a cluster point process model, this command returns a `function(x,y)` giving the two-dimensional probability density of the cluster offspring points assuming a cluster parent located at the origin.

Value

A function in the R\ language with arguments `x,y,...`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

Rolf Turner `<r.turner@auckland.ac.nz>`

and Ege Rubak `<rubak@math.aau.dk>`

See Also

`clusterfield`, `kppm`

Examples

```
fit <- kppm(redwood ~ x, "MatClust")
f <- clusterkernel(fit)
f(0.1, 0.2)
```

clusterradius	*Compute or Extract Effective Range of Cluster Kernel*

Description

Given a cluster point process model, this command returns a value beyond which the the probability density of the cluster offspring is neglible.

Usage

```
clusterradius(model, ...)

## S3 method for class 'kppm'
clusterradius(model, ..., thresh = NULL, precision = FALSE)

## S3 method for class 'character'
clusterradius(model, ..., thresh = NULL, precision = FALSE)
```

Arguments

`model`	Cluster model. Either a fitted cluster or Cox model (object of class `"kppm"`), or a character string specifying the type of cluster model.
`...`	Parameter values for the model, when `model` is a character string.
`thresh`	Numerical threshold relative to the cluster kernel value at the origin (parent location) determining when the cluster kernel will be considered neglible. A sensible default is provided.
`precision`	Logical. If `precision=TRUE` the precision of the calculated range is returned as an attribute to the range. See details.

Details

Given a cluster model this function by default returns the effective range of the model with the given parameters as used in spatstat. For the Matern cluster model (see e.g. `rMatClust`) this is simply the finite radius of the offsring density given by the paramter `scale` irrespective of other options given to this function. The remaining models in spatstat have infinite theoretical range, and an effective finite value is given as follows: For the Thomas model (see e.g. `rThomas` the default is `4*scale` where scale is the scale or standard deviation parameter of the model. If `thresh` is given the value is instead found as described for the other models below.

For the Cauchy model (see e.g. `rCauchy`) and the Variance Gamma (Bessel) model (see e.g. `rVarGamma`) the value of `thresh` defaults to 0.001, and then this is used to compute the range numerically as follows. If $k(x,y) = k_0(r)$ with $r = \sqrt(x^2 + y^2)$ denotes the isotropic cluster kernel then $f(r) = 2\pi r k_0(r)$ is the density function of the offspring distance from the parent. The range is determined as the value of r where $f(r)$ falls below `thresh` times $k_0(r)$.

If `precision=TRUE` the precision related to the chosen range is returned as an attribute. Here the precision is defined as the polar integral of the kernel from distance 0 to the calculated range. Ideally this should be close to the value 1 which would be obtained for the true theretical infinite range.

Value

A positive numeric.

Additionally, the precision related to this range value is returned as an attribute `"prec"`, if `precision=TRUE`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

clusterkernel, kppm, rMatClust, rThomas, rCauchy, rVarGamma, rNeymanScott.

Examples

```
fit <- kppm(redwood ~ x, "MatClust")
clusterradius(fit)

clusterradius("Thomas", scale = .1)
clusterradius("Thomas", scale = .1, thresh = 0.001)
clusterradius("VarGamma", scale = .1, nu = 2, precision = TRUE)
```

---

clusterset *Allard-Fraley Estimator of Cluster Feature*

Description

Detect high-density features in a spatial point pattern using the (unrestricted) Allard-Fraley estimator.

Usage

```
clusterset(X, what=c("marks", "domain"),
           ..., verbose=TRUE,
           fast=FALSE,
           exact=!fast)
```

Arguments

X	A dimensional spatial point pattern (object of class "ppp").
what	Character string or character vector specifying the type of result. See Details.
verbose	Logical value indicating whether to print progress reports.
fast	Logical. If FALSE (the default), the Dirichlet tile areas will be computed exactly using polygonal geometry, so that the optimal choice of tiles will be computed exactly. If TRUE, the Dirichlet tile areas will be approximated using pixel counting, so the optimal choice will be approximate.
exact	Logical. If TRUE, the Allard-Fraley estimator of the domain will be computed exactly using polygonal geometry. If FALSE, the Allard-Fraley estimator of the domain will be approximated by a binary pixel mask. The default is initially set to FALSE.
...	Optional arguments passed to as.mask to control the pixel resolution if exact=FALSE.

Details

Allard and Fraley (1997) developed a technique for recognising features of high density in a spatial point pattern in the presence of random clutter.

This algorithm computes the *unrestricted* Allard-Fraley estimator. The Dirichlet (Voronoi) tessellation of the point pattern `X` is computed. The smallest `m` Dirichlet cells are selected, where the number `m` is determined by a maximum likelihood criterion.

- If `fast=FALSE` (the default), the areas of the tiles of the Dirichlet tessellation will be computed exactly using polygonal geometry. This ensures that the optimal selection of tiles is computed exactly.
- If `fast=TRUE`, the Dirichlet tile areas will be approximated by counting pixels. This is faster, and is usually correct (depending on the pixel resolution, which is controlled by the arguments `...`).

The type of result depends on the character vector `what`.

- If `what="marks"` the result is the point pattern `X` with a vector of marks labelling each point with a value `yes` or `no` depending on whether the corresponding Dirichlet cell is selected by the Allard-Fraley estimator. In other words each point of `X` is labelled as either a cluster point or a non-cluster point.
- If `what="domain"`, the result is the Allard-Fraley estimator of the cluster feature set, which is the union of all the selected Dirichlet cells, represented as a window (object of class `"owin"`).
- If `what=c("marks", "domain")` the result is a list containing both of the results described above.

Computation of the Allard-Fraley set estimator depends on the argument `exact`.

- If `exact=TRUE` (the default), the Allard-Fraley set estimator will be computed exactly using polygonal geometry. The result is a polygonal window.
- If `exact=FALSE`, the Allard-Fraley set estimator will be approximated by a binary pixel mask. This is faster than the exact computation. The result is a binary mask.

Value

If `what="marks"`, a multitype point pattern (object of class `"ppp"`).

If `what="domain"`, a window (object of class `"owin"`).

If `what=c("marks", "domain")` (the default), a list consisting of a multitype point pattern and a window.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Allard, D. and Fraley, C. (1997) Nonparametric maximum likelihood estimation of features in spatial point processes using Voronoi tessellation. *Journal of the American Statistical Association* **92**, 1485–1493.

See Also

`nnclean`, `sharpen`

Examples

```
opa <- par(mfrow=c(1,2))
W <- grow.rectangle(as.rectangle(letterR), 1)
X <- superimpose(runifpoint(300, letterR),
                 runifpoint(50, W), W=W)
plot(W, main="clusterset(X, 'm')")
plot(clusterset(X, "marks", fast=TRUE), add=TRUE, chars=c(1, 3), cols=1:2)
plot(letterR, add=TRUE)
plot(W, main="clusterset(X, 'd')")
plot(clusterset(X, "domain", exact=FALSE), add=TRUE)
plot(letterR, add=TRUE)
par(opa)
```

`coef.mppm` *Coefficients of Point Process Model Fitted to Multiple Point Patterns*

Description

Given a point process model fitted to a list of point patterns, extract the coefficients of the fitted model. A method for `coef`.

Usage

```
## S3 method for class 'mppm'
coef(object, ...)
```

Arguments

`object`	The fitted point process model (an object of class "mppm")
`...`	Ignored.

Details

This function is a method for the generic function `coef`.

The argument `object` must be a fitted point process model (object of class "mppm") produced by the fitting algorithm `mppm`). This represents a point process model that has been fitted to a list of several point pattern datasets. See `mppm` for information.

This function extracts the vector of coefficients of the fitted model. This is the estimate of the parameter vector θ such that the conditional intensity of the model is of the form

$$\lambda(u, x) = \exp(\theta S(u, x))$$

where $S(u, x)$ is a (vector-valued) statistic.

For example, if the model `object` is the uniform Poisson process, then `coef(object)` will yield a single value (named "(Intercept)") which is the logarithm of the fitted intensity of the Poisson process.

If the fitted model includes random effects (i.e. if the argument `random` was specified in the call to `mppm`), then the fitted coefficients are different for each point pattern in the original data, so `coef(object)` is a data frame with one row for each point pattern, and one column for each parameter. Use `fixef.mppm` to extract the vector of fixed effect coefficients, and `ranef.mppm` to extract the random effect coefficients at each level.

Use `print.mppm` to print a more useful description of the fitted model.

Value

Either a vector containing the fitted coefficients, or a data frame containing the fitted coefficients for each point pattern.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented in **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

See Also

`fixef.mppm` and `ranef.mppm` for the fixed and random effect coefficients in a model that includes random effects.

`print.mppm`, `mppm`

Examples

```
H <- hyperframe(X=waterstriders)

fit.Poisson <- mppm(X ~ 1, H)
coef(fit.Poisson)

# The single entry "(Intercept)"
# is the log of the fitted intensity of the Poisson process

fit.Strauss <- mppm(X~1, H, Strauss(7))
coef(fit.Strauss)

# The two entries "(Intercept)" and "Interaction"
# are respectively log(beta) and log(gamma)
# in the usual notation for Strauss(beta, gamma, r)

# Tweak data to exaggerate differences
H$X[[1]] <- rthin(H$X[[1]], 0.3)
# Model with random effects
fitran <- mppm(X ~ 1, H, random=~1|id)
coef(fitran)
```

`coef.ppm` *Coefficients of Fitted Point Process Model*

Description

Given a point process model fitted to a point pattern, extract the coefficients of the fitted model. A method for `coef`.

Usage

```
## S3 method for class 'ppm'
coef(object, ...)
```

Arguments

`object`	The fitted point process model (an object of class "ppm")
`...`	Ignored.

Details

This function is a method for the generic function `coef`.

The argument `object` must be a fitted point process model (object of class "ppm"). Such objects are produced by the maximum pseudolikelihood fitting algorithm `ppm`).

This function extracts the vector of coefficients of the fitted model. This is the estimate of the parameter vector θ such that the conditional intensity of the model is of the form

$$\lambda(u, x) = \exp(\theta S(u, x))$$

where $S(u, x)$ is a (vector-valued) statistic.

For example, if the model `object` is the uniform Poisson process, then `coef(object)` will yield a single value (named "(Intercept)") which is the logarithm of the fitted intensity of the Poisson process.

Use `print.ppm` to print a more useful description of the fitted model.

Value

A vector containing the fitted coefficients.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`print.ppm`, `ppm.object`, `ppm`

Examples

```
data(cells)

poi <- ppm(cells, ~1, Poisson())
coef(poi)
# This is the log of the fitted intensity of the Poisson process

stra <- ppm(cells, ~1, Strauss(r=0.07))
coef(stra)

# The two entries "(Intercept)" and "Interaction"
# are respectively log(beta) and log(gamma)
# in the usual notation for Strauss(beta, gamma, r)
```

coef.slrm *Coefficients of Fitted Spatial Logistic Regression Model*

Description

Extracts the coefficients (parameters) from a fitted Spatial Logistic Regression model.

Usage

```
## S3 method for class 'slrm'
coef(object, ...)
```

Arguments

`object`	a fitted spatial logistic regression model. An object of class `"slrm"`.
`...`	Ignored.

Details

This is a method for `coef` for fitted spatial logistic regression models (objects of class `"slrm"`, usually obtained from the function `slrm`).

It extracts the fitted canonical parameters, i.e.\ the coefficients in the linear predictor of the spatial logistic regression.

Value

Numeric vector of coefficients.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` `<adrian@maths.uwa.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`slrm`

Examples

```
X <- rpoispp(42)
fit <- slrm(X ~ x+y)
coef(fit)
```

collapse.fv *Collapse Several Function Tables into One*

Description

Combines several function tables (objects of class "fv") into a single function table, merging columns that are identical and relabelling columns that are different.

Usage

```
## S3 method for class 'fv'
collapse(object, ..., same = NULL, different = NULL)

## S3 method for class 'anylist'
collapse(object, ..., same = NULL, different = NULL)
```

Arguments

object	An object of class "fv", or a list of such objects.
...	Additional objects of class "fv".
same	Character string or character vector specifying a column or columns, present in each "fv" object, that are identical in each object. This column or columns will be included only once.
different	Character string or character vector specifying a column or columns, present in each "fv" object, that contain different values in each object. Each of these columns of data will be included, with labels that distinguish them from each other.

Details

This is a method for the generic function collapse.

It combines the data in several function tables (objects of class "fv", see fv.object) to make a single function table. It is essentially a smart wrapper for cbind.fv.

A typical application is to calculate the same summary statistic (such as the K function) for different point patterns, and then to use collapse.fv to combine the results into a single object that can easily be plotted. See the Examples.

The arguments object and ... should be function tables (objects of class "fv", see fv.object) that are compatible in the sense that they have the same values of the function argument.

The argument same identifies any columns that are present in each function table, and which are known to contain exactly the same values in each table. This column or columns will be included only once in the result.

The argument different identifies any columns that are present in each function table, and which contain different numerical values in each table. Each of these columns will be included, with labels to distinguish them.

Columns that are not named in same or different will not be included.

Value

Object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

fv.object, cbind.fv

Examples

```
# generate simulated data
X <- replicate(3, rpoispp(100), simplify=FALSE)
names(X) <- paste("Simulation", 1:3)
# compute K function estimates
Klist <- anylapply(X, Kest)
# collapse
K <- collapse(Klist, same="theo", different="iso")
K
```

---

colourmap *Colour Lookup Tables*

Description

Create a colour map (colour lookup table).

Usage

```
colourmap(col, ..., range=NULL, breaks=NULL, inputs=NULL, gamma=1)
```

Arguments

col	Vector of values specifying colours
...	Ignored.
range	Interval to be mapped. A numeric vector of length 2, specifying the endpoints of the range of values to be mapped. Incompatible with breaks or inputs.
inputs	Values to which the colours are associated. A factor or vector of the same length as col. Incompatible with breaks or range.
breaks	Breakpoints for the colour map. A numeric vector of length equal to length(col)+1. Incompatible with range or inputs.
gamma	Exponent for the gamma correction, when range is given. A single positive number. See Details.

Details

A colour map is a mechanism for associating colours with data. It can be regarded as a function, mapping data to colours.

The command `colourmap` creates an object representing a colour map, which can then be used to control the plot commands in the **spatstat** package. It can also be used to compute the colour assigned to any data value.

The argument `col` specifies the colours to which data values will be mapped. It should be a vector whose entries can be interpreted as colours by the standard R graphics system. The entries can be string names of colours like `"red"`, or integers that refer to colours in the standard palette, or strings containing six-letter hexadecimal codes like `"#F0A0FF"`.

Exactly one of the arguments `range`, `inputs` or `breaks` must be specified by name.

If `inputs` is given, then it should be a vector or factor, of the same length as `col`. The entries of `inputs` can be any atomic type (e.g. numeric, logical, character, complex) or factor values. The resulting colour map associates the value `inputs[i]` with the colour `col[i]`.

If `range` is given, then it determines the interval of the real number line that will be mapped. It should be a numeric vector of length 2. The interval will be divided evenly into bands, each of which is assigned one of the colours in `col`. (If `gamma` is given, then the bands are equally spaced on a scale where the original values are raised to the power `gamma`.)

If `breaks` is given, then it determines the precise intervals of the real number line which are mapped to each colour. It should be a numeric vector, of length at least 2, with entries that are in increasing order. Infinite values are allowed. Any number in the range between `breaks[i]` and `breaks[i+1]` will be mapped to the colour `col[i]`.

The result is an object of class `"colourmap"`. There are `print` and `plot` methods for this class. Some plot commands in the **spatstat** package accept an object of this class as a specification of the colour map.

The result is also a function `f` which can be used to compute the colour assigned to any data value. That is, `f(x)` returns the character value of the colour assigned to `x`. This also works for vectors of data values.

Value

A function, which is also an object of class `"colourmap"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

The plot method `plot.colourmap`.

See the R help file on `colours` for information about the colours that R recognises, and how to manipulate them.

To make a smooth transition between colours, see `interp.colourmap`. To alter individual colour values, see `tweak.colourmap`. To extract or replace all colour values, see `colouroutputs`.

See `colourtools` for more tools to manipulate colour values.

See `lut` for lookup tables.

Examples

```
# colour map for real numbers, using breakpoints
cr <- colourmap(c("red", "blue", "green"), breaks=c(0,5,10,15))
cr
cr(3.2)
cr(c(3,5,7))
# a large colour map
co <- colourmap(rainbow(100), range=c(-1,1))
co(0.2)
# colour map for discrete set of values
ct <- colourmap(c("red", "green"), inputs=c(FALSE, TRUE))
ct(TRUE)
```

---

colouroutputs *Extract or Assign Colour Values in a Colour Map*

Description

Extract the colour values in a colour map, or assign new colour values.

Usage

```
colouroutputs(x)

colouroutputs(x) <- value
```

Arguments

x	A colour map (object of class "colourmap").
value	A vector of values that can be interpreted as colours.

Details

An object of class "colourmap" is effectively a function that maps its inputs (numbers or factor levels) to colour values.

The command `colouroutputs(x)` extracts the colour values in the colour map `x`.

The assignment `colouroutputs(x) <- value` replaces the colour values in the colour map `x` by the entries in `value`. The replacement vector `value` should have the same length as `colouroutputs(x)`, and its entries should be interpretable as colours.

To change only some of the colour values in a colour map, it may be easier to use `tweak.colourmap`.

Value

The result of `colouroutputs` is a character vector of colour values. The result of the assignment `colouroutputs(x) <- value` is another colour map (object of class "colourmap").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

colourmap, interp.colourmap, tweak.colourmap, colourtools.

Examples

```
m <- colourmap(rainbow(5), range=c(0,1))
m
# reverse order of colours
colouroutputs(m) <- rev(colouroutputs(m))
m
```

---

colourtools — *Convert and Compare Colours in Different Formats*

Description

These functions convert between different formats for specifying a colour in R, determine whether colours are equivalent, and convert colour to greyscale.

Usage

```
col2hex(x)
rgb2hex(v, maxColorValue=255)
rgb2hsva(red, green=NULL, blue=NULL, alpha=NULL, maxColorValue=255)
paletteindex(x)
samecolour(x,y)
complementarycolour(x)
interp.colours(x, length.out=512)
is.colour(x)
to.grey(x, weights=c(0.299, 0.587, 0.114), transparent=FALSE)
is.grey(x)
to.opaque(x)
to.transparent(x, fraction)
to.saturated(x, s=1)
```

Arguments

`x,y` Any valid specification for a colour or sequence of colours accepted by col2rgb.

`v` A numeric vector of length 3, giving the RGB values of a single colour, or a 3-column matrix giving the RGB values of several colours. Alternatively a vector of length 4 or a matrix with 4 columns, giving the RGB and alpha (transparency) values.

`red,green,blue,alpha` Arguments acceptable to rgb determining the red, green, blue channels and optionally the alpha (transparency) channel. Note that `red` can also be a matrix with 3 **rows** giving the RGB values, or a matrix with 4 rows giving RGB and alpha values.

`maxColorValue` Number giving the maximum possible value for the entries in `v` or `red,green,blue,alpha`.

`weights` Numeric vector of length 3 giving relative weights for the red, green, and blue channels respectively.

`transparent`	Logical value indicating whether transparent colours should be converted to transparent grey values (`transparent=TRUE`) or converted to opaque grey values (`transparent=FALSE`, the default).
`fraction`	Transparency fraction. Numerical value or vector of values between 0 and 1, giving the opaqueness of a colour. A fully opaque colour has `fraction=1`.
`length.out`	Integer. Length of desired sequence.
`s`	Saturation value (between 0 and 1).

Details

`is.colour(x)` can be applied to any kind of data `x` and returns `TRUE` if `x` can be interpreted as a colour or colours. The remaining functions expect data that can be interpreted as colours.

`col2hex` converts colours specified in any format into their hexadecimal character codes.

`rgb2hex` converts RGB colour values into their hexadecimal character codes. It is a very minor extension to `rgb`. Arguments to `rgb2hex` should be similar to arguments to `rgb`.

`rgb2hsva` converts RGB colour values into HSV colour values including the alpha (transparency) channel. It is an extension of `rgb2hsv`. Arguments to `rgb2hsva` should be similar to arguments to `rgb2hsv`.

`paletteindex` checks whether the colour or colours specified by `x` are available in the default palette returned by `palette()`. If so, it returns the index or indices of the colours in the palette. If not, it returns `NA`.

`samecolour` decides whether two colours `x` and `y` are equivalent.

`is.grey` determines whether each entry of `x` is a greyscale colour, and returns a logical vector.

`to.grey` converts the colour data in `x` to greyscale colours. Alternatively `x` can be an object of class `"colourmap"` and `to.grey(x)` is the modified colour map.

`to.opaque` converts the colours in `x` to opaque (non-transparent) colours, and `to.transparent` converts them to transparent colours with a specified transparency value. Note that `to.transparent(x,1)` is equivalent to `to.opaque(x)`.

For `to.grey`, `to.opaque` and `to.transparent`, if all the data in `x` specifies colours from the standard palette, and if the result would be equivalent to `x`, then the result is identical to `x`.

`to.saturated` converts each colour in `x` to its fully-saturated equivalent. For example, pink is mapped to red. Shades of grey are converted to black; white is unchanged.

`complementarycolour` replaces each colour by its complementary colour in RGB space (the colour obtained by replacing RGB values `(r, g, b)` by `(255-r, 255-g, 255-b)`). The transparency value is not changed. Alternatively `x` can be an object of class `"colourmap"` and `complementarycolour(x)` is the modified colour map.

`interp.colours` interpolates between each successive pair of colours in a sequence of colours, to generate a more finely-spaced sequence. It uses linear interpolation in HSV space (with hue represented as a two-dimensional unit vector).

Value

For `col2hex` and `rgb2hex` a character vector containing hexadecimal colour codes.

For `to.grey`, `to.opaque` and `to.transparent`, either a character vector containing hexadecimal colour codes, or a value identical to the input `x`.

For `rgb2hsva`, a matrix with 3 or 4 rows containing HSV colour values.

For `paletteindex`, an integer vector, possibly containing `NA` values.

For `samecolour` and `is.grey`, a logical value or logical vector.

Warning

`paletteindex("green")` returns `NA` because the green colour in the default palette is called `"green3"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`col2rgb`, `rgb2hsv`, `palette`.

See also the class of colour map objects in the **spatstat** package: `colourmap`, `interp.colourmap`, `tweak.colourmap`.

Examples

```
samecolour("grey", "gray")
paletteindex("grey")
col2hex("orange")
to.grey("orange")
to.saturated("orange")
complementarycolour("orange")
is.grey("lightgrey")
is.grey(8)
to.transparent("orange", 0.5)
to.opaque("red")
interp.colours(c("orange", "red", "violet"), 5)
```

`commonGrid` *Determine A Common Spatial Domain And Pixel Resolution*

Description

Determine a common spatial domain and pixel resolution for several spatial objects such as images, masks, windows and point patterns.

Usage

```
commonGrid(...)
```

Arguments

`...`	Any number of pixel images (objects of class `"im"`), binary masks (objects of class `"owin"` of type `"mask"`) or data which can be converted to binary masks by `as.mask`.

Details

This function determines a common spatial resolution and spatial domain for several spatial objects.

The arguments `...` may be pixel images, binary masks, or other spatial objects acceptable to `as.mask`.

The common pixel grid is determined by inspecting all the pixel images and binary masks in the argument list, finding the pixel grid with the highest spatial resolution, and extending this pixel grid to cover the bounding box of all the spatial objects.

The return value is a binary mask `M`, representing the bounding box at the chosen pixel resolution. Use `as.im(X, W=M)` to convert a pixel image `X` to this new pixel resolution. Use `as.mask(W, xy=M)` to convert a window `W` to a binary mask at this new pixel resolution. See the Examples.

Value

A binary mask (object of class `"owin"` and type `"mask"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`harmonise.im`, `compatible.im`, `as.im`

Examples

```
A <- setcov(square(1))
G <- density(runifpoint(42), dimyx=16)
H <- commonGrid(A, letterR, G)
newR <- as.mask(letterR, xy=H)
newG <- as.im(G, W=H)
```

---

compareFit *Residual Diagnostics for Multiple Fitted Models*

Description

Compares several fitted point process models using the same residual diagnostic.

Usage

```
compareFit(object, Fun, r = NULL, breaks = NULL, ...,
         trend = ~1, interaction = Poisson(), rbord = NULL,
         modelnames = NULL, same = NULL, different = NULL)
```

Arguments

object	Object or objects to be analysed. Either a fitted point process model (object of class "ppm"), a point pattern (object of class "ppp"), or a list of these objects.
Fun	Diagnostic function to be computed for each model. One of the functions Kcom, Kres, Gcom, Gres, psst, psstA or psstG or a string containing one of these names.
r	Optional. Vector of values of the argument r at which the diagnostic should be computed. This argument is usually not specified. There is a sensible default.
breaks	Optional alternative to r for advanced use.
...	Extra arguments passed to Fun.
trend,interaction,rbord	Optional. Arguments passed to ppm to fit a point process model to the data, if object is a point pattern or list of point patterns. See ppm for details. Each of these arguments can be a list, specifying different trend, interaction and/or rbord values to be used to generate different fitted models.
modelnames	Character vector. Short descriptive names for the different models.
same,different	Character strings or character vectors passed to collapse.fv to determine the format of the output.

Details

This is a convenient way to collect diagnostic information for several different point process models fitted to the same point pattern dataset, or for point process models of the same form fitted to several different datasets, etc.

The first argument, object, is usually a list of fitted point process models (objects of class "ppm"), obtained from the model-fitting function ppm.

For convenience, object can also be a list of point patterns (objects of class "ppp"). In that case, point process models will be fitted to each of the point pattern datasets, by calling ppm using the arguments trend (for the first order trend), interaction (for the interpoint interaction) and rbord (for the erosion distance in the border correction for the pseudolikelihood). See ppm for details of these arguments.

Alternatively object can be a single point pattern (object of class "ppp") and one or more of the arguments trend, interaction or rbord can be a list. In this case, point process models will be fitted to the same point pattern dataset, using each of the model specifications listed.

The diagnostic function Fun will be applied to each of the point process models. The results will be collected into a single function value table. The modelnames are used to label the results from each fitted model.

Value

Function value table (object of class "fv").

Author(s)

Ege Rubak <rubak@math.aau.dk>, Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Jesper Møller.

See Also

ppm, Kcom, Kres, Gcom, Gres, psst, psstA, psstG, collapse.fv

Examples

```
nd <- 40

ilist <- list(Poisson(), Geyer(7, 2), Strauss(7))
iname <- c("Poisson", "Geyer", "Strauss")

K <- compareFit(swedishpines, Kcom, interaction=ilist, rbord=9,
         correction="translate",
         same="trans", different="tcom", modelnames=iname, nd=nd)
K
```

---

compatible — *Test Whether Objects Are Compatible*

Description

Tests whether two or more objects of the same class are compatible.

Usage

```
compatible(A, B, ...)
```

Arguments

`A,B,...`	Two or more objects of the same class

Details

This generic function is used to check whether the objects `A` and `B` (and any additional objects `...`) are compatible.

What is meant by 'compatible' depends on the class of object.

There are methods for the classes `"fv"`, `"fasp"`, `"im"` and `"unitname"`.

Value

Logical value: `TRUE` if the objects are compatible, and `FALSE` if they are not.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`compatible.fv`, `compatible.fasp`, `compatible.im`, `compatible.unitname` `compatible.fasp` *Test Whether Function Arrays Are Compatible*

Description

Tests whether two or more function arrays (class `"fasp"`) are compatible.

Usage

```
## S3 method for class 'fasp'
compatible(A, B, ...)
```

Arguments

`A,B,...`	Two or more function arrays (object of class `"fasp"`).

Details

An object of class `"fasp"` can be regarded as an array of functions. Such objects are returned by the command `alltypes`.

This command tests whether such objects are compatible (so that, for example, they could be added or subtracted). It is a method for the generic command `compatible`.

The function arrays are compatible if the arrays have the same dimensions, and the corresponding elements in each cell of the array are compatible as defined by `compatible.fv`.

Value

Logical value: `TRUE` if the objects are compatible, and `FALSE` if they are not.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`eval.fasp`

`compatible.fv` *Test Whether Function Objects Are Compatible*

Description

Tests whether two or more function objects (class `"fv"`) are compatible.

Usage

```
## S3 method for class 'fv'
compatible(A, B, ...)
```

Arguments

`A,B,...`	Two or more function value objects (class `"fv"`).

Details

An object of class `"fv"` is essentially a data frame containing several different statistical estimates of the same function. Such objects are returned by `Kest` and its relatives.

This command tests whether such objects are compatible (so that, for example, they could be added or subtracted). It is a method for the generic command `compatible`.

The functions are compatible if they have been evaluated at the same sequence of values of the argument `r`, and if the statistical estimates have the same names.

Value

Logical value: `TRUE` if the objects are compatible, and `FALSE` if they are not.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`eval.fv`

---

compatible.im *Test Whether Pixel Images Are Compatible*

Description

Tests whether two or more pixel image objects have compatible dimensions.

Usage

```
 ## S3 method for class 'im'
compatible(A, B, ..., tol=1e-6)
```

Arguments

`A,B,...`	Two or more pixel images (objects of class `"im"`).
`tol`	Tolerance factor

Details

This function tests whether the pixel images `A` and `B` (and any additional images `...`) have compatible pixel dimensions. They are compatible if they have the same number of rows and columns, the same physical pixel dimensions, and occupy the same rectangle in the plane.

The argument `tol` specifies the maximum tolerated error in the pixel coordinates, expressed as a fraction of the dimensions of a single pixel.

Value

Logical value: TRUE if the images are compatible, and FALSE if they are not.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

eval.im, harmonise.im, commonGrid

---

compileK *Generic Calculation of K Function and Pair Correlation Function*

Description

Low-level functions which calculate the estimated K function and estimated pair correlation function (or any similar functions) from a matrix of pairwise distances and optional weights.

Usage

```
compileK(D, r, weights = NULL, denom = 1,
         check = TRUE, ratio = FALSE, fname = "K")

compilepcf(D, r, weights = NULL, denom = 1,
         check = TRUE, endcorrect = TRUE, ratio=FALSE,
  ..., fname = "g")
```

Arguments

D	A square matrix giving the distances between all pairs of points.
r	An equally spaced, finely spaced sequence of distance values.
weights	Optional numerical weights for the pairwise distances. A numeric matrix with the same dimensions as D. If absent, the weights are taken to equal 1.
denom	Denominator for the estimator. A single number, or a numeric vector with the same length as r. See Details.
check	Logical value specifying whether to check that D is a valid matrix of pairwise distances.
ratio	Logical value indicating whether to store ratio information. See Details.
...	Optional arguments passed to density.default controlling the kernel smoothing.
endcorrect	Logical value indicating whether to apply End Correction of the pair correlation estimate at r=0.
fname	Character string giving the name of the function being estimated.

Details

These low-level functions construct estimates of the K function or pair correlation function, or any similar functions, given only the matrix of pairwise distances and optional weights associated with these distances.

These functions are useful for code development and for teaching, because they perform a common task, and do the housekeeping required to make an object of class "fv" that represents the estimated function. However, they are not very efficient.

compileK calculates the weighted estimate of the K function,

$$\hat{K}(r) = (1/v(r)) \sum_i \sum_j 1\{d_{ij} \le r\} w_{ij}$$

and compilepcf calculates the weighted estimate of the pair correlation function,

$$\hat{g}(r) = (1/v(r)) \sum_i \sum_j \kappa(d_{ij} - r) w_{ij}$$

where d_{ij} is the distance between spatial points i and j, with corresponding weight w_{ij}, and $v(r)$ is a specified denominator. Here κ is a fixed-bandwidth smoothing kernel.

For a point pattern in two dimensions, the usual denominator $v(r)$ is constant for the K function, and proportional to r for the pair correlation function. See the Examples.

The result is an object of class "fv" representing the estimated function. This object has only one column of function values. Additional columns (such as a column giving the theoretical value) must be added by the user, with the aid of bind.fv.

If ratio=TRUE, the result also belongs to class "rat" and has attributes containing the numerator and denominator of the function estimate. This allows function estimates from several datasets to be pooled using pool.

Value

An object of class "fv" representing the estimated function.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

Kest, pcf for definitions of the K function and pair correlation function.

bind.fv to add more columns.

Examples

```
X <- japanesepines
D <- pairdist(X)
Wt <- edge.Ripley(X, D)
lambda <- intensity(X)
a <- (npoints(X)-1) * lambda
r <- seq(0, 0.25, by=0.01)
K <- compileK(D=D, r=r, weights=Wt, denom=a)
g <- compilepcf(D=D, r=r, weights=Wt, denom= a * 2 * pi * r)
```

complement.owin *Take Complement of a Window*

Description

Take the set complement of a window, within its enclosing rectangle or in a larger rectangle.

Usage

```
complement.owin(w, frame=as.rectangle(w))
```

Arguments

w	an object of class "owin" describing a window of observation for a point pattern.
frame	Optional. The enclosing rectangle, with respect to which the set complement is taken.

Details

This yields a window object (of class "owin", see owin.object) representing the set complement of w with respect to the rectangle frame.

By default, frame is the enclosing box of w (originally specified by the arguments xrange and yrange given to owin when w was created). If frame is specified, it must be a rectangle (an object of class "owin" whose type is "rectangle") and it must be larger than the enclosing box of w. This rectangle becomes the enclosing box for the resulting window.

If w is a rectangle, then frame must be specified. Otherwise an error will occur (since the complement of w in itself is empty).

For rectangular and polygonal windows, the complement is computed by reversing the sign of each boundary polygon, while for binary masks it is computed by negating the pixel values.

Value

Another object of class "owin" representing the complement of the window, i.e. the inside of the window becomes the outside.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin, owin.object

Examples

```
# rectangular
a <- owin(c(0,1),c(0,1))
b <- owin(c(-1,2),c(-1,2))
bmina <- complement.owin(a, frame=b)
# polygonal
data(demopat)
w <- Window(demopat)
outside <- complement.owin(w)
# mask
w <- as.mask(Window(demopat))
outside <- complement.owin(w)
```

---

concatxy — *Concatenate x,y Coordinate Vectors*

Description

Concatenate any number of pairs of `x` and `y` coordinate vectors.

Usage

```
concatxy(...)
```

Arguments

`...`	Any number of arguments, each of which is a structure containing elements `x` and `y`.

Details

This function can be used to superimpose two or more point patterns of unmarked points (but see also `superimpose` which is recommended).

It assumes that each of the arguments in `...` is a structure containing (at least) the elements `x` and `y`. It concatenates all the `x` elements into a vector `x`, and similarly for `y`, and returns these concatenated vectors.

Value

A list with two components `x` and `y`, which are the concatenations of all the corresponding `x` and `y` vectors in the argument list.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`superimpose`, `quadscheme`

Examples

```
dat <- runifrect(30)
xy <- list(x=runif(10),y=runif(10))
new <- concatxy(dat, xy)
```

Concom *The Connected Component Process Model*

Description

Creates an instance of the Connected Component point process model which can then be fitted to point pattern data.

Usage

```
Concom(r)
```

Arguments

r	Threshold distance

Details

This function defines the interpoint interaction structure of a point process called the connected component process. It can be used to fit this model to point pattern data.

The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the connected component interaction is yielded by the function Concom(). See the examples below.

In **standard form**, the connected component process (Baddeley and Møller, 1989) with disc radius r, intensity parameter κ and interaction parameter γ is a point process with probability density

$$f(x_1, \ldots, x_n) = \alpha \kappa^{n(x)} \gamma^{-C(x)}$$

for a point pattern x, where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, and $C(x)$ is defined below. Here α is a normalising constant.

To define the term C(x), suppose that we construct a planar graph by drawing an edge between each pair of points x_i, x_j which are less than r units apart. Two points belong to the same connected component of this graph if they are joined by a path in the graph. Then $C(x)$ is the number of connected components of the graph.

The interaction parameter γ can be any positive number. If $\gamma = 1$ then the model reduces to a Poisson process with intensity κ. If $\gamma < 1$ then the process is regular, while if $\gamma > 1$ the process is clustered. Thus, a connected-component interaction process can be used to model either clustered or regular point patterns.

In **spatstat**, the model is parametrised in a different form, which is easier to interpret. In **canonical form**, the probability density is rewritten as

$$f(x_1, \ldots, x_n) = \alpha \beta^{n(x)} \gamma^{-U(x)}$$

where β is the new intensity parameter and $U(x) = C(x) - n(x)$ is the interaction potential. In this formulation, each isolated point of the pattern contributes a factor β to the probability density (so the first order trend is β). The quantity $U(x)$ is a true interaction potential, in the sense that $U(x) = 0$ if the point pattern x does not contain any points that lie close together.

When a new point u is added to an existing point pattern x, the rescaled potential $-U(x)$ increases by zero or a positive integer. The increase is zero if u is not close to any point of x. The increase is a positive integer k if there are k different connected components of x that lie close to u. Addition of the point u contributes a factor $\beta\eta^\delta$ to the probability density, where δ is the increase in potential.

If desired, the original parameter κ can be recovered from the canonical parameter by $\kappa = \beta\gamma$.

The *nonstationary* connected component process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location, rather than a constant beta.

Note the only argument of `Concom()` is the threshold distance `r`. When `r` is fixed, the model becomes an exponential family. The canonical parameters $\log(\beta)$ and $\log(\gamma)$ are estimated by `ppm()`, not fixed in `Concom()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the connected component process with disc radius r.

Edge correction

The interaction distance of this process is infinite. There are no well-established procedures for edge correction for fitting such models, and accordingly the model-fitting function `ppm` will give an error message saying that the user must specify an edge correction. A reasonable solution is to use the border correction at the same distance `r`, as shown in the Examples.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A.J. and Møller, J. (1989) Nearest-neighbour Markov point processes and random sets. *International Statistical Review* **57**, 89–121.

See Also

`ppm`, `pairwise.family`, `ppm.object`

Examples

```
# prints a sensible description of itself
Concom(r=0.1)

# Fit the stationary connected component process to redwood data
ppm(redwood, ~1, Concom(r=0.07), rbord=0.07)

# Fit the stationary connected component process to `cells' data
ppm(cells, ~1, Concom(r=0.06), rbord=0.06)
# eta=0 indicates hard core process.

# Fit a nonstationary connected component model
# with log-cubic polynomial trend
## Not run:
```

```
  ppm(swedishpines, ~polynom(x/10,y/10,3), Concom(r=7), rbord=7)

## End(Not run)
```

---

connected *Connected components*

Description

Finds the topologically-connected components of a spatial object, such as the connected clumps of pixels in a binary image.

Usage

```
connected(X, ...)

## S3 method for class 'owin'
connected(X, ..., method="C")

## S3 method for class 'im'
connected(X, ..., background = NA, method="C")
```

Arguments

X	A spatial object such as a pixel image (object of class "im") or a window (object of class "owin").
background	Optional. Treat pixels with this value as being part of the background.
method	String indicating the algorithm to be used. Either "C" or "interpreted". See Details.
...	Arguments passed to as.mask to determine the pixel resolution.

Details

The function connected is generic, with methods for pixel images (class "im") and windows (class "owin") described here. There is also a method for point patterns described in connected.ppp.

The functions described here compute the connected component transform (Rosenfeld and Pfalz, 1966) of a binary image or binary mask. The argument X is first converted into a pixel image with logical values. Then the algorithm identifies the connected components (topologically-connected clumps of pixels) in the foreground.

Two pixels belong to the same connected component if they have the value TRUE and if they are neighbours (in the 8-connected sense). This rule is applied repeatedly until it terminates. Then each connected component contains all the pixels that can be reached by stepping from neighbour to neighbour.

If method="C", the computation is performed by a compiled C language implementation of the classical algorithm of Rosenfeld and Pfalz (1966). If method="interpreted", the computation is performed by an R implementation of the algorithm of Park et al (2000).

The result is a factor-valued image, with levels that correspond to the connected components. The Examples show how to extract each connected component as a separate window object.

Value

A pixel image (object of class "im") with factor values. The levels of the factor correspond to the connected components.

Warnings

It may be hard to distinguish different components in the default plot because the colours of nearby components may be very similar. See the Examples for a randomised colour map.

The algorithm for method="interpreted" can be very slow for large images (or images where the connected components include a large number of pixels).

Author(s)

Original R code by Julian Burgos, University of Washington. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

References

Park, J.-M., Looney, C.G. and Chen, H.-C. (2000) Fast connected component labeling algorithm using a divide and conquer technique. Pages 373-376 in S.Y. Shin (ed) *Computers and Their Applications:* Proceedings of the ISCA 15th International Conference on Computers and Their Applications, March 29-31, 2000, New Orleans, Louisiana USA. ISCA 2000, ISBN 1-880843-32-3.

Rosenfeld, A. and Pfalz, J.L. (1966) Sequential operations in digital processing. *Journal of the Association for Computing Machinery* **13** 471-494.

See Also

connected.ppp, im.object, tess

Examples

```
d <- distmap(cells, dimyx=256)
X <- levelset(d, 0.07)
plot(X)
Z <- connected(X)
plot(Z)
# or equivalently
Z <- connected(d <= 0.07)

# number of components
nc <- length(levels(Z))
# plot with randomised colour map
plot(Z, col=hsv(h=sample(seq(0,1,length=nc), nc)))

# how to extract the components as a list of windows
W <- tiles(tess(image=Z))
```

connected.linnet *Connected Components of a Linear Network*

Description

Find the topologically-connected components of a linear network.

Usage

```
## S3 method for class 'linnet'
connected(X, ..., what = c("labels", "components"))
```

Arguments

X	A linear network (object of class "linnet").
...	Ignored.
what	Character string specifying the kind of result.

Details

The function connected is generic. This is the method for linear networks (objects of class "linnet").

Two vertices of the network are connected if they are joined by a path in the network. This function divides the network into subsets, such that all points in a subset are connected to each other.

If what="labels" the return value is a factor with one entry for each vertex of X, identifying which connected component the vertex belongs to.

If what="components" the return value is a list of linear networks, which are the connected components of X.

Value

If what="labels", a factor. If what="components", a list of linear networks.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Suman Rakshit.

See Also

thinNetwork

Examples

```
# remove some edges from a network to make it disconnected
plot(simplenet, col="grey", main="", lty=2)
A <- thinNetwork(simplenet, retainedges=-c(3,5))
plot(A, add=TRUE, lwd=2)
# find the connected components
connected(A)
cA <- connected(A, what="components")
plot(cA[[1]], add=TRUE, col="green", lwd=2)
plot(cA[[2]], add=TRUE, col="blue", lwd=2)
```

`connected.lpp` *Connected Components of a Point Pattern on a Linear Network*

Description

Finds the topologically-connected components of a point pattern on a linear network, when all pairs of points closer than a threshold distance are joined.

Usage

```
## S3 method for class 'lpp'
connected(X, R=Inf, ..., dismantle=TRUE)
```

Arguments

`X`	A linear network (object of class "lpp").
`R`	Threshold distance. Pairs of points will be joined together if they are closer than R units apart, measured by the shortest path in the network. The default R=Inf implies that points will be joined together if they are mutually connected by any path in the network.
`dismantle`	Logical. If TRUE (the default), the network itself will be divided into its path-connected components using connected.linnet.
`...`	Ignored.

Details

The function connected is generic. This is the method for point patterns on a linear network (objects of class "lpp"). It divides the point pattern X into one or more groups of points.

If R=Inf (the default), then X is divided into groups such that any pair of points in the same group can be joined by a path in the network.

If R is a finite number, then two points of X are declared to be *R-close* if they lie closer than R units apart, measured by the length of the shortest path in the network. Two points are *R-connected* if they can be reached by a series of steps between R-close pairs of points of X. Then X is divided into groups such that any pair of points in the same group is R-connected.

If dismantle=TRUE (the default) the algorithm first checks whether the network is connected (i.e. whether any pair of vertices can be joined by a path in the network), and if not, the network is decomposed into its connected components.

Value

A point pattern (of class "lpp") with marks indicating the grouping, or a list of such point patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

thinNetwork

Examples

```
# remove some edges from a network to make it disconnected
plot(simplenet, col="grey", main="", lty=2)
A <- thinNetwork(simplenet, retainedges=-c(3,5))
plot(A, add=TRUE, lwd=2)
X <- runiflpp(10, A)
# find the connected components
cX <- connected(X)
plot(cX[[1]], add=TRUE, col="blue", lwd=2)
```

---

connected.ppp *Connected Components of a Point Pattern*

Description

Finds the topologically-connected components of a point pattern, when all pairs of points closer than a threshold distance are joined.

Usage

```
## S3 method for class 'ppp'
connected(X, R, ...)

## S3 method for class 'pp3'
connected(X, R, ...)
```

Arguments

`X`	A point pattern (object of class "ppp" or "pp3").
`R`	Threshold distance. Pairs of points closer than `R` units apart will be joined together.
`...`	Other arguments, not recognised by these methods.

Details

This function can be used to identify clumps of points in a point pattern.

The function `connected` is generic. This file documents the methods for point patterns in dimension two or three (objects of class "ppp" or "pp3").

The point pattern `X` is first converted into an abstract graph by joining every pair of points that lie closer than `R` units apart. Then the connected components of this graph are identified.

Two points in `X` belong to the same connected component if they can be reached by a series of steps between points of `X`, each step being shorter than `R` units in length.

The result is a vector of labels for the points of `X` where all the points in a connected component have the same label.

Value

A point pattern, equivalent to `X` except that the points have factor-valued marks, with levels corresponding to the connected components.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

connected.im, im.object, tess

Examples

```
Y <- connected(redwoodfull, 0.1)
if(interactive()) {
 plot(Y, cols=1:length(levels(marks(Y))),
      main="connected(redwoodfull, 0.1)")
}
X <- osteo$pts[[1]]
Z <- connected(X, 32)
if(interactive()) {
 plot(Z, col=marks(Z), main="")
}
```

---

contour.im *Contour plot of pixel image*

Description

Generates a contour plot of a pixel image.

Usage

```
## S3 method for class 'im'
contour(x, ..., main,
             axes=FALSE, add=FALSE, col=par("fg"),
             clipwin=NULL, show.all=!add, do.plot=TRUE)
```

Arguments

x	Pixel image to be plotted. An object of class "im".
main	Character string to be displayed as the main title.
axes	Logical. If TRUE, coordinate axes are plotted (with tick marks) around a region slightly larger than the image window. If FALSE (the default), no axes are plotted, and a box is drawn tightly around the image window. Ignored if add=TRUE.
add	Logical. If FALSE, a new plot is created. If TRUE, the contours are drawn over the existing plot.
col	Colour in which to draw the contour lines. Either a single value that can be interpreted as a colour value, or a colourmap object.
clipwin	Optional. A window (object of class "owin"). Only this subset of the data will be displayed.
...	Other arguments passed to contour.default controlling the contour plot; see Details.

show.all Logical value indicating whether to display all plot elements including the main title, bounding box, and (if axis=TRUE) coordinate axis markings. Default is TRUE for new plots and FALSE for added plots.

do.plot Logical value indicating whether to actually perform the plot.

Details

This is a method for the generic contour function, for objects of the class "im".

An object of class "im" represents a pixel image; see im.object.

This function displays the values of the pixel image x as a contour plot on the current plot device, using equal scales on the x and y axes.

The appearance of the plot can be modified using any of the arguments listed in the help for contour.default. Useful ones include:

nlevels Number of contour levels to plot.

drawlabels Whether to label the contour lines with text.

col,lty,lwd Colour, type, and width of contour lines.

See contour.default for a full list of these arguments.

The defaults for any of the abovementioned arguments can be reset using spatstat.options("par.contour").

If col is a colour map (object of class "colourmap", see colourmap) then the contours will be plotted in different colours as determined by the colour map. The contour at level z will be plotted in the colour col(z) associated with this level in the colour map.

Value

none.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

im.object, plot.im, persp.im

Examples

```
# an image
Z <- setcov(owin())
contour(Z, axes=TRUE)
contour(Z)

co <- colourmap(rainbow(100), range=c(0,1))
contour(Z, col=co, lwd=2)
```

contour.imlist *Array of Contour Plots*

Description

Generates an array of contour plots.

Usage

```
## S3 method for class 'imlist'
contour(x, ...)

## S3 method for class 'listof'
contour(x, ...)
```

Arguments

x	An object of the class "imlist" representing a list of pixel images. Alternatively x may belong to the outdated class "listof".
...	Arguments passed to plot.solist to control the spatial arrangement of panels, and arguments passed to contour.im to control the display of each panel.

Details

This is a method for the generic command contour for the class "imlist". An object of class "imlist" represents a list of pixel images.

(The outdated class "listof" is also handled.)

Each entry in the list x will be displayed as a contour plot, in an array of panels laid out on the same graphics display, using plot.solist. Invididual panels are plotted by contour.im.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

plot.solist, contour.im

Examples

```
# Multitype point pattern
 contour(D <- density(split(amacrine)))
```

convexhull *Convex Hull*

Description

Computes the convex hull of a spatial object.

Usage

```
convexhull(x)
```

Arguments

x	a window (object of class "owin"), a point pattern (object of class "ppp"), a line segment pattern (object of class "psp"), or an object that can be converted to a window by as.owin.

Details

This function computes the convex hull of the spatial object x.

Value

A window (an object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin, convexhull.xy, is.convex

Examples

```
data(demopat)
W <- Window(demopat)
plot(convexhull(W), col="lightblue", border=NA)
plot(W, add=TRUE, lwd=2)
```

`convexhull.xy` *Convex Hull of Points*

Description

Computes the convex hull of a set of points in two dimensions.

Usage

```
convexhull.xy(x, y=NULL)
```

Arguments

`x`	vector of `x` coordinates of observed points, or a 2-column matrix giving `x,y` coordinates, or a list with components `x,y` giving coordinates (such as a point pattern object of class `"ppp"`.)
`y`	(optional) vector of `y` coordinates of observed points, if `x` is a vector.

Details

Given an observed pattern of points with coordinates given by `x` and `y`, this function computes the convex hull of the points, and returns it as a window.

Value

A window (an object of class `"owin"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`owin`, `as.owin`, `convexhull`, `bounding.box.xy`, `ripras`

Examples

```
x <- runif(30)
y <- runif(30)
w <- convexhull.xy(x,y)
plot(owin(), main="convexhull.xy(x,y)", lty=2)
plot(w, add=TRUE)
points(x,y)

X <- rpoispp(30)
plot(X, main="convexhull.xy(X)")
plot(convexhull.xy(X), add=TRUE)
```

convexify *Weil's Convexifying Operation*

Description

Converts the window W into a convex set by rearranging the edges, preserving spatial orientation of each edge.

Usage

```
convexify(W, eps)
```

Arguments

W	A window (object of class "owin").
eps	Optional. Minimum edge length of polygonal approximation, if W is not a polygon.

Details

Weil (1995) defined a convexification operation for windows W that belong to the convex ring (that is, for any W which is a finite union of convex sets). Note that this is **not** the same as the convex hull.

The convexified set $f(W)$ has the same total boundary length as W and the same distribution of orientations of the boundary. If W is a polygonal set, then the convexification $f(W)$ is obtained by rearranging all the edges of W in order of their spatial orientation.

The argument W must be a window. If it is not already a polygonal window, it is first converted to one, using simplify.owin. The edges are sorted in increasing order of angular orientation and reassembled into a convex polygon.

Value

A window (object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Weil, W. (1995) The estimation of mean particle shape and mean particle number in overlapping particle systems in the plane. *Advances in Applied Probability* **27**, 102–119.

See Also

convexhull for the convex hull of a window.

Examples

```
opa <- par(mfrow=c(1,2))
plot(letterR)
plot(convexify(letterR))
par(opa)
```

`convolve.im` *Convolution of Pixel Images*

Description

Computes the convolution of two pixel images.

Usage

```
convolve.im(X, Y=X, ..., reflectX=FALSE, reflectY=FALSE)
```

Arguments

`X`	A pixel image (object of class `"im"`.
`Y`	Optional. Another pixel image.
`...`	Ignored.
`reflectX,reflectY`	Logical values specifying whether the images `X` and `Y` (respectively) should be reflected in the origin before computing the convolution.

Details

The *convolution* of two pixel images X and Y in the plane is the function $C(v)$ defined for each vector v as

$$C(v) = \int X(u)Y(v-u)\,\mathrm{d}u$$

where the integral is over all spatial locations u, and where $X(u)$ and $Y(u)$ denote the pixel values of X and Y respectively at location u.

This command computes a discretised approximation to the convolution, using the Fast Fourier Transform. The return value is another pixel image (object of class `"im"`) whose greyscale values are values of the convolution.

If `reflectX = TRUE` then the pixel image `X` is reflected in the origin (see `reflect`) before the convolution is computed, so that `convolve.im(X,Y,reflectX=TRUE)` is mathematically equivalent to `convolve.im(reflect(X), Y)`. (These two commands are not exactly equivalent, because the reflection is performed in the Fourier domain in the first command, and reflection is performed in the spatial domain in the second command).

Similarly if `reflectY = TRUE` then the pixel image `Y` is reflected in the origin before the convolution is computed, so that `convolve.im(X,Y,reflectY=TRUE)` is mathematically equivalent to `convolve.im(X, reflect(Y))`.

Value

A pixel image (an object of class `"im"`) representing the convolution of `X` and `Y`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

imcov, reflect

Examples

```
X <- as.im(letterR)
Y <- as.im(square(1))
plot(convolve.im(X, Y))
plot(convolve.im(X, Y, reflectX=TRUE))
plot(convolve.im(X))
```

---

coords *Extract or Change Coordinates of a Spatial or Spatiotemporal Point Pattern*

Description

Given any kind of spatial or space-time point pattern, this function extracts the (space and/or time and/or local) coordinates of the points and returns them as a data frame.

Usage

```
  coords(x, ...)
  ## S3 method for class 'ppp'
coords(x, ...)
  ## S3 method for class 'ppx'
coords(x, ..., spatial = TRUE, temporal = TRUE, local=TRUE)
  coords(x, ...) <- value
  ## S3 replacement method for class 'ppp'
coords(x, ...) <- value
  ## S3 replacement method for class 'ppx'
coords(x, ..., spatial = TRUE, temporal = TRUE, local=TRUE) <- value
```

Arguments

x A point pattern: either a two-dimensional point pattern (object of class "ppp"), a three-dimensional point pattern (object of class "pp3"), or a general multidimensional space-time point pattern (object of class "ppx").

... Further arguments passed to methods.

spatial,temporal,local
Logical values indicating whether to extract spatial, temporal and local coordinates, respectively. The default is to return all such coordinates. (Only relevant to ppx objects).

value New values of the coordinates. A numeric vector with one entry for each point in x, or a numeric matrix or data frame with one row for each point in x.

Details

The function coords extracts the coordinates from a point pattern. The function coords<- replaces the coordinates of the point pattern with new values.

Both functions coords and coords<- are generic, with methods for the classes "ppp") and "ppx". An object of class "pp3" also inherits from "ppx" and is handled by the method for "ppx".

Value

coords returns a data.frame with one row for each point, containing the coordinates. coords<- returns the altered point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppx, pp3, ppp, as.hyperframe.ppx, as.data.frame.ppx.

Examples

```
df <- data.frame(x=runif(4),y=runif(4),t=runif(4))
X <- ppx(data=df, coord.type=c("s","s","t"))
coords(X)
coords(X, temporal=FALSE)
coords(X) <- matrix(runif(12), ncol=3)
```

---

corners *Corners of a rectangle*

Description

Returns the four corners of a rectangle

Usage

```
corners(window)
```

Arguments

window	A window. An object of class owin, or data in any format acceptable to as.owin().

Details

This trivial function is occasionally convenient. If window is of type "rectangle" this returns the four corners of the window itself; otherwise, it returns the corners of the bounding rectangle of the window.

Value

A list with two components x and y, which are numeric vectors of length 4 giving the coordinates of the four corner points of the (bounding rectangle of the) window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quad.object, quadscheme

Examples

```
w <- unit.square()
corners(w)
     # returns list(x=c(0,1,0,1),y=c(0,0,1,1))
```

---

covering *Cover Region with Discs*

Description

Given a spatial region, this function finds an efficient covering of the region using discs of a chosen radius.

Usage

```
covering(W, r, ..., giveup=1000)
```

Arguments

W	A window (object of class "owin").
r	positive number: the radius of the covering discs.
...	extra arguments passed to as.mask controlling the pixel resolution for the calculations.
giveup	Maximum number of attempts to place additional discs.

Details

This function finds an efficient covering of the window W using discs of the given radius r. The result is a point pattern giving the centres of the discs.

The algorithm tries to use as few discs as possible, but is not guaranteed to find the minimal number of discs. It begins by placing a hexagonal grid of points inside W, then adds further points until every location inside W lies no more than r units away from one of the points.

Value

A point pattern (object of class "ppp") giving the centres of the discs.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Examples

```
rr <- 0.5
X <- covering(letterR, rr)
plot(grow.rectangle(Frame(X), rr), type="n", main="")
plot(X, pch=16, add=TRUE, col="red")
plot(letterR, add=TRUE, lwd=3)
plot(X %mark% (2*rr), add=TRUE, markscale=1)
```

---

crossdist *Pairwise distances*

Description

Computes the distances between pairs of 'things' taken from two different datasets.

Usage

```
crossdist(X, Y, ...)
```

Arguments

X,Y	Two objects of the same class.
...	Additional arguments depending on the method.

Details

Given two datasets X and Y (representing either two point patterns or two line segment patterns) crossdist computes the Euclidean distance from each thing in the first dataset to each thing in the second dataset, and returns a matrix containing these distances.

The function crossdist is generic, with methods for point patterns (objects of class "ppp"), line segment patterns (objects of class "psp"), and a default method. See the documentation for crossdist.ppp, crossdist.psp or crossdist.default for further details.

Value

A matrix whose [i,j] entry is the distance from the i-th thing in the first dataset to the j-th thing in the second dataset.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

crossdist.ppp, crossdist.psp, crossdist.default, pairdist, nndist crossdist.default *Pairwise distances between two different sets of points*

Description

Computes the distances between each pair of points taken from two different sets of points.

Usage

```
## Default S3 method:
crossdist(X, Y, x2, y2, ...,
               period=NULL, method="C", squared=FALSE)
```

Arguments

X,Y	Numeric vectors of equal length specifying the coordinates of the first set of points.
x2,y2	Numeric vectors of equal length specifying the coordinates of the second set of points.
...	Ignored.
period	Optional. Dimensions for periodic edge correction.
method	String specifying which method of calculation to use. Values are "C" and "interpreted".
squared	Logical. If squared=TRUE, the squared distances are returned instead (this computation is faster).

Details

Given two sets of points, this function computes the Euclidean distance from each point in the first set to each point in the second set, and returns a matrix containing these distances.

This is a method for the generic function crossdist.

This function expects X and Y to be numeric vectors of equal length specifying the coordinates of the first set of points. The arguments x2,y2 specify the coordinates of the second set of points.

Alternatively if period is given, then the distances will be computed in the 'periodic' sense (also known as 'torus' distance). The points will be treated as if they are in a rectangle of width period[1] and height period[2]. Opposite edges of the rectangle are regarded as equivalent.

The argument method is not normally used. It is retained only for checking the validity of the software. If method = "interpreted" then the distances are computed using interpreted R code only. If method="C" (the default) then C code is used. The C code is faster by a factor of 4.

Value

A matrix whose [i,j] entry is the distance from the i-th point in the first set of points to the j-th point in the second set of points.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

crossdist, crossdist.ppp, crossdist.psp, pairdist, nndist, Gest

Examples

```
d <- crossdist(runif(7), runif(7), runif(12), runif(12))
d <- crossdist(runif(7), runif(7), runif(12), runif(12), period=c(1,1))
```

---

crossdist.lpp *Pairwise distances between two point patterns on a linear network*

Description

Computes the distances between pairs of points taken from two different point patterns on the same linear network.

Usage

```
  ## S3 method for class 'lpp'
crossdist(X, Y, ..., method="C")
```

Arguments

X,Y	Point patterns on a linear network (objects of class "lpp"). They must lie on the *same* network.
...	Ignored.
method	String specifying which method of calculation to use. Values are "C" and "interpreted".

Details

Given two point patterns on a linear network, this function computes the Euclidean distance from each point in the first pattern to each point in the second pattern, measuring distance by the shortest path in the network.

This is a method for the generic function crossdist for point patterns on a linear network (objects of class "lpp").

This function expects two point pattern objects X and Y on the *same* linear network, and returns the matrix whose [i,j] entry is the shortest-path distance from X[i] to Y[j].

The argument method is not normally used. It is retained only for checking the validity of the software. If method = "interpreted" then the distances are computed using interpreted R code only. If method="C" (the default) then C code is used. The C code is much faster.

If two points cannot be joined by a path, the distance between them is infinite (Inf).

Value

A matrix whose [i,j] entry is the distance from the i-th point in X to the j-th point in Y. Matrix entries are nonnegative numbers or infinity (Inf).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

crossdist, crossdist.ppp, pairdist, nndist

Examples

```
v <- split(chicago)
X <- v$cartheft
Y <- v$burglary
d <- crossdist(X, Y)
```

---

crossdist.pp3 *Pairwise distances between two different three-dimensional point patterns*

Description

Computes the distances between pairs of points taken from two different three-dimensional point patterns.

Usage

```
  ## S3 method for class 'pp3'
crossdist(X, Y, ..., periodic=FALSE, squared=FALSE)
```

Arguments

X,Y	Point patterns in three dimensions (objects of class "pp3").
...	Ignored.
periodic	Logical. Specifies whether to apply a periodic edge correction.
squared	Logical. If squared=TRUE, the squared distances are returned instead (this computation is faster).

Details

Given two point patterns in three-dimensional space, this function computes the Euclidean distance from each point in the first pattern to each point in the second pattern, and returns a matrix containing these distances.

This is a method for the generic function crossdist for three-dimensional point patterns (objects of class "pp3").

This function expects two point patterns X and Y, and returns the matrix whose [i,j] entry is the distance from X[i] to Y[j].

Alternatively if periodic=TRUE, then provided the windows containing X and Y are identical and are rectangular, then the distances will be computed in the 'periodic' sense (also known as 'torus' distance): opposite edges of the rectangle are regarded as equivalent. This is meaningless if the window is not a rectangle.

Value

A matrix whose [i,j] entry is the distance from the i-th point in X to the j-th point in Y.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

based on code for two dimensions by Pavel Grabarnik.

See Also

crossdist, pairdist, nndist, G3est

Examples

```
X <- runifpoint3(20)
Y <- runifpoint3(30)
d <- crossdist(X, Y)
d <- crossdist(X, Y, periodic=TRUE)
```

---

crossdist.ppp *Pairwise distances between two different point patterns*

Description

Computes the distances between pairs of points taken from two different point patterns.

Usage

```
  ## S3 method for class 'ppp'
crossdist(X, Y, ..., periodic=FALSE, method="C", squared=FALSE)
```

Arguments

X,Y	Point patterns (objects of class "ppp").
...	Ignored.
periodic	Logical. Specifies whether to apply a periodic edge correction.
method	String specifying which method of calculation to use. Values are "C" and "interpreted".
squared	Logical. If squared=TRUE, the squared distances are returned instead (this computation is faster).

Details

Given two point patterns, this function computes the Euclidean distance from each point in the first pattern to each point in the second pattern, and returns a matrix containing these distances.

This is a method for the generic function crossdist for point patterns (objects of class "ppp").

This function expects two point patterns X and Y, and returns the matrix whose [i,j] entry is the distance from X[i] to Y[j].

Alternatively if periodic=TRUE, then provided the windows containing X and Y are identical and are rectangular, then the distances will be computed in the 'periodic' sense (also known as 'torus' distance): opposite edges of the rectangle are regarded as equivalent. This is meaningless if the window is not a rectangle.

The argument method is not normally used. It is retained only for checking the validity of the software. If method = "interpreted" then the distances are computed using interpreted R code only. If method="C" (the default) then C code is used. The C code is faster by a factor of 4.

Value

A matrix whose `[i,j]` entry is the distance from the `i`-th point in `X` to the `j`-th point in `Y`.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

`crossdist`, `crossdist.default`, `crossdist.psp`, `pairdist`, `nndist`, `Gest`

Examples

```
data(cells)
d <- crossdist(cells, runifpoint(6))
d <- crossdist(cells, runifpoint(6), periodic=TRUE)
```

`crossdist.ppx` *Pairwise Distances Between Two Different Multi-Dimensional Point Patterns*

Description

Computes the distances between pairs of points taken from two different multi-dimensional point patterns.

Usage

```
  ## S3 method for class 'ppx'
crossdist(X, Y, ...)
```

Arguments

`X,Y`	Multi-dimensional point patterns (objects of class `"ppx"`).
`...`	Arguments passed to `coords.ppx` to determine which coordinates should be used.

Details

Given two point patterns in multi-dimensional space, this function computes the Euclidean distance from each point in the first pattern to each point in the second pattern, and returns a matrix containing these distances.

This is a method for the generic function `crossdist` for three-dimensional point patterns (objects of class `"ppx"`).

This function expects two multidimensional point patterns `X` and `Y`, and returns the matrix whose `[i,j]` entry is the distance from `X[i]` to `Y[j]`.

By default, both spatial and temporal coordinates are extracted. To obtain the spatial distance between points in a space-time point pattern, set `temporal=FALSE`.

Value

A matrix whose [i,j] entry is the distance from the i-th point in X to the j-th point in Y.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

crossdist, pairdist, nndist

Examples

```
df <- data.frame(x=runif(3),y=runif(3),z=runif(3),w=runif(3))
X <- ppx(data=df)
df <- data.frame(x=runif(5),y=runif(5),z=runif(5),w=runif(5))
Y <- ppx(data=df)
d <- crossdist(X, Y)
```

---

crossdist.psp *Pairwise distances between two different line segment patterns*

Description

Computes the distances between all pairs of line segments taken from two different line segment patterns.

Usage

```
  ## S3 method for class 'psp'
crossdist(X, Y, ..., method="C", type="Hausdorff")
```

Arguments

X,Y	Line segment patterns (objects of class "psp").
...	Ignored.
method	String specifying which method of calculation to use. Values are "C" and "interpreted". Usually not specified.
type	Type of distance to be computed. Options are "Hausdorff" and "separation". Partial matching is used.

Details

This is a method for the generic function crossdist.

Given two line segment patterns, this function computes the distance from each line segment in the first pattern to each line segment in the second pattern, and returns a matrix containing these distances.

The distances between line segments are measured in one of two ways:

- if type="Hausdorff", distances are computed in the Hausdorff metric. The Hausdorff distance between two line segments is the *maximum* distance from any point on one of the segments to the nearest point on the other segment.
- if type="separation", distances are computed as the *minimum* distance from a point on one line segment to a point on the other line segment. For example, line segments which cross over each other have separation zero.

The argument method is not normally used. It is retained only for checking the validity of the software. If method = "interpreted" then the distances are computed using interpreted R code only. If method="C" (the default) then compiled C code is used. The C code is several times faster.

Value

A matrix whose [i,j] entry is the distance from the i-th line segment in X to the j-th line segment in Y.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

pairdist, nndist, Gest

Examples

```
L1 <- psp(runif(5), runif(5), runif(5), runif(5), owin())
L2 <- psp(runif(10), runif(10), runif(10), runif(10), owin())
D <- crossdist(L1, L2)
#result is a 5 x 10 matrix
S <- crossdist(L1, L2, type="sep")
```

---

crossing.linnet *Crossing Points between Linear Network and Other Lines*

Description

Find all the crossing-points between a linear network and another pattern of lines or line segments.

Usage

```
crossing.linnet(X, Y)
```

Arguments

X	Linear network (object of class "linnet").
Y	A linear network, or a spatial pattern of line segments (class "psp") or infinite lines (class "infline").

Details

All crossing-points between X and Y are determined. The result is a point pattern on the network X.

Value

Point pattern on a linear network (object of class "lpp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

crossing.psp

Examples

```
plot(simplenet, main="")
L <- infline(p=runif(3), theta=runif(3, max=pi/2))
plot(L, col="red")
Y <- crossing.linnet(simplenet, L)
plot(Y, add=TRUE, cols="blue")
```

---

crossing.psp *Crossing Points of Two Line Segment Patterns*

Description

Finds any crossing points between two line segment patterns.

Usage

```
crossing.psp(A,B,fatal=TRUE,details=FALSE)
```

Arguments

A,B	Line segment patterns (objects of class "psp").
details	Logical value indicating whether to return additional information. See below.
fatal	Logical value indicating what to do if the windows of A and B do not overlap. See Details.

Details

This function finds any crossing points between the line segment patterns A and B.

A crossing point occurs whenever one of the line segments in A intersects one of the line segments in B, at a nonzero angle of intersection.

The result is a point pattern consisting of all the intersection points.

If details=TRUE, additional information is computed, specifying where each intersection point came from. The resulting point pattern has a data frame of marks, with columns named iA, jB, tA, tB. The marks iA and jB are the indices of the line segments in A and B, respectively, which produced each intersection point. The marks tA and tB are numbers between 0 and 1 specifying the position of the intersection point along the original segments.

If the windows Window(A) and Window(B) do not overlap, then an error will be reported if fatal=TRUE, while if fatal=FALSE an error will not occur and the result will be NULL.

Value

Point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

selfcrossing.psp, psp.object, ppp.object.

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
b <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(a, col="green", main="crossing.psp")
plot(b, add=TRUE, col="blue")
P <- crossing.psp(a,b)
plot(P, add=TRUE, col="red")
as.data.frame(crossing.psp(a,b,details=TRUE))
```

---

cut.im — *Convert Pixel Image from Numeric to Factor*

Description

Transform the values of a pixel image from numeric values into a factor.

Usage

```
  ## S3 method for class 'im'
cut(x, ...)
```

Arguments

x	A pixel image. An object of class "im".
...	Arguments passed to cut.default. They determine the breakpoints for the mapping from numerical values to factor values. See cut.default.

Details

This simple function applies the generic cut operation to the pixel values of the image x. The range of pixel values is divided into several intervals, and each interval is associated with a level of a factor. The result is another pixel image, with the same window and pixel grid as x, but with the numeric value of each pixel discretised by replacing it by the factor level.

This function is a convenient way to inspect an image and to obtain summary statistics. See the examples.

To select a subset of an image, use the subset operator [.im instead.

Value

A pixel image (object of class "im") with pixel values that are a factor. See im.object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

cut, im.object

Examples

```
# artificial image data
Z <- setcov(square(1))

Y <- cut(Z, 3)
Y <- cut(Z, breaks=seq(0,1,length=5))

# cut at the quartiles
# (divides the image into 4 equal areas)
Y <- cut(Z, quantile(Z))
```

---

cut.lpp *Classify Points in a Point Pattern on a Network*

Description

For a point pattern on a linear network, classify the points into distinct types according to the numerical marks in the pattern, or according to another variable.

Usage

```
  ## S3 method for class 'lpp'
cut(x, z=marks(x), ...)
```

Arguments

x	A point pattern on a linear network (object of class "lpp").
z	Data determining the classification. A numeric vector, a factor, a pixel image on a linear network (class "linim"), a function on a linear network (class "linfun"), a tessellation on a linear network (class "lintess"), a string giving the name of a column of marks, or one of the coordinate names "x", "y", "seg" or "tp".
...	Arguments passed to cut.default. They determine the breakpoints for the mapping from numerical values in z to factor values in the output. See cut.default.

Details

This function has the effect of classifying each point in the point pattern x into one of several possible types. The classification is based on the dataset z, which may be either

- a factor (of length equal to the number of points in z) determining the classification of each point in x. Levels of the factor determine the classification.

- a numeric vector (of length equal to the number of points in z). The range of values of z will be divided into bands (the number of bands is determined by ...) and z will be converted to a factor using cut.default.
- a pixel image on a network (object of class "linim"). The value of z at each point of x will be used as the classifying variable.
- a function on a network (object of class "linfun", see linfun). The value of z at each point of x will be used as the classifying variable.
- a tessellation on a network (object of class "lintess", see lintess). Each point of x will be classified according to the tile of the tessellation into which it falls.
- a character string, giving the name of one of the columns of marks(x), if this is a data frame.
- a character string identifying one of the coordinates: the spatial coordinates "x", "y" or the segment identifier "seg" or the fractional coordinate along the segment, "tp".

The default is to take z to be the vector of marks in x (or the first column in the data frame of marks of x, if it is a data frame). If the marks are numeric, then the range of values of the numerical marks is divided into several intervals, and each interval is associated with a level of a factor. The result is a marked point pattern, on the same linear network, with the same point locations as x, but with the numeric mark of each point discretised by replacing it by the factor level. This is a convenient way to transform a marked point pattern which has numeric marks into a multitype point pattern, for example to plot it or analyse it. See the examples.

To select some points from x, use the subset operators [.lpp or subset.lpp instead.

Value

A multitype point pattern on the same linear network, that is, a point pattern object (of class "lpp") with a marks vector that is a factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

cut, lpp, lintess, linfun, linim

Examples

```
X <- runiflpp(20, simplenet)
f <- linfun(function(x,y,seg,tp) { x }, simplenet)
plot(cut(X, f, breaks=4))
plot(cut(X, "x", breaks=4))
plot(cut(X, "seg"))
```

cut.ppp *Classify Points in a Point Pattern*

Description

Classifies the points in a point pattern into distinct types according to the numerical marks in the pattern, or according to another variable.

Usage

```
## S3 method for class 'ppp'
cut(x, z=marks(x), ...)
```

Arguments

x	A two-dimensional point pattern. An object of class "ppp".
z	Data determining the classification. A numeric vector, a factor, a pixel image, a window, a tessellation, or a string giving the name of a column of marks or the name of a spatial coordinate.
...	Arguments passed to cut.default. They determine the breakpoints for the mapping from numerical values in z to factor values in the output. See cut.default.

Details

This function has the effect of classifying each point in the point pattern x into one of several possible types. The classification is based on the dataset z, which may be either

- a factor (of length equal to the number of points in z) determining the classification of each point in x. Levels of the factor determine the classification.
- a numeric vector (of length equal to the number of points in z). The range of values of z will be divided into bands (the number of bands is determined by ...) and z will be converted to a factor using cut.default.
- a pixel image (object of class "im"). The value of z at each point of x will be used as the classifying variable.
- a tessellation (object of class "tess", see tess). Each point of x will be classified according to the tile of the tessellation into which it falls.
- a window (object of class "owin"). Each point of x will be classified according to whether it falls inside or outside this window.
- a character string, giving the name of one of the columns of marks(x), if this is a data frame.
- a character string "x" or "y" identifying one of the spatial coordinates.

The default is to take z to be the vector of marks in x (or the first column in the data frame of marks of x, if it is a data frame). If the marks are numeric, then the range of values of the numerical marks is divided into several intervals, and each interval is associated with a level of a factor. The result is a marked point pattern, with the same window and point locations as x, but with the numeric mark of each point discretised by replacing it by the factor level. This is a convenient way to transform a marked point pattern which has numeric marks into a multitype point pattern, for example to plot it or analyse it. See the examples.

To select some points from a point pattern, use the subset operators [.ppp or subset.ppp instead.

Value

A multitype point pattern, that is, a point pattern object (of class "ppp") with a marks vector that is a factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

cut, ppp.object, tess

Examples

```
 # (1) cutting based on numeric marks of point pattern

 trees <- longleaf
 # Longleaf Pines data
 # the marks are positive real numbers indicating tree diameters.


 ## Not run:
 plot(trees)

## End(Not run)

 # cut the range of tree diameters into three intervals
 long3 <- cut(trees, breaks=3)
 ## Not run:
 plot(long3)

## End(Not run)

 # adult trees defined to have diameter at least 30 cm
 long2 <- cut(trees, breaks=c(0,30,100), labels=c("Sapling", "Adult"))
 plot(long2)
 plot(long2, cols=c("green","blue"))

 # (2) cutting based on another numeric vector
 # Divide Swedish Pines data into 3 classes
 # according to nearest neighbour distance

 swedishpines
 plot(cut(swedishpines, nndist(swedishpines), breaks=3))

 # (3) cutting based on tessellation
 # Divide Swedish Pines study region into a 4 x 4 grid of rectangles
 # and classify points accordingly

 tes <- tess(xgrid=seq(0,96,length=5),ygrid=seq(0,100,length=5))
 plot(cut(swedishpines, tes))
 plot(tes, lty=2, add=TRUE)

 # (4) multivariate marks
 finpines
```

```
cut(finpines, "height", breaks=4)
```

data.ppm *Extract Original Data from a Fitted Point Process Model*

Description

Given a fitted point process model, this function extracts the original point pattern dataset to which the model was fitted.

Usage

```
data.ppm(object)
```

Arguments

object	fitted point process model (an object of class "ppm").

Details

An object of class "ppm" represents a point process model that has been fitted to data. It is typically produced by the model-fitting algorithm ppm. The object contains complete information about the original data point pattern to which the model was fitted. This function extracts the original data pattern.

See ppm.object for a list of all operations that can be performed on objects of class "ppm".

Value

A point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm.object, ppp.object

Examples

```
data(cells)
fit <- ppm(cells, ~1, Strauss(r=0.1))
X <- data.ppm(fit)
# 'X' is identical to 'cells'
```

`dclf.progress` *Progress Plot of Test of Spatial Pattern*

Description

Generates a progress plot (envelope representation) of the Diggle-Cressie-Loosmore-Ford test or the Maximum Absolute Deviation test for a spatial point pattern.

Usage

```
dclf.progress(X, ...)
mad.progress(X, ...)
mctest.progress(X, fun = Lest, ...,
                exponent = 1, nrank = 1,
                interpolate = FALSE, alpha, rmin=0)
```

Arguments

`X`	Either a point pattern (object of class `"ppp"`, `"lpp"` or other class), a fitted point process model (object of class `"ppm"`, `"kppm"` or other class) or an envelope object (class `"envelope"`).
`...`	Arguments passed to `mctest.progress` or to `envelope`. Useful arguments include `fun` to determine the summary function, `nsim` to specify the number of Monte Carlo simulations, `alternative` to specify one-sided or two-sided envelopes, and `verbose=FALSE` to turn off the messages.
`fun`	Function that computes the desired summary statistic for a point pattern.
`exponent`	Positive number. The exponent of the L^p distance. See Details.
`nrank`	Integer. The rank of the critical value of the Monte Carlo test, amongst the `nsim` simulated values. A rank of 1 means that the minimum and maximum simulated values will become the critical values for the test.
`interpolate`	Logical value indicating how to compute the critical value. If `interpolate=FALSE` (the default), a standard Monte Carlo test is performed, and the critical value is the largest simulated value of the test statistic (if `nrank=1`) or the `nrank`-th largest (if `nrank` is another number). If `interpolate=TRUE`, kernel density estimation is applied to the simulated values, and the critical value is the upper `alpha` quantile of this estimated distribution.
`alpha`	Optional. The significance level of the test. Equivalent to `nrank/(nsim+1)` where `nsim` is the number of simulations.
`rmin`	Optional. Left endpoint for the interval of r values on which the test statistic is calculated.

Details

The Diggle-Cressie-Loosmore-Ford test and the Maximum Absolute Deviation test for a spatial point pattern are described in `dclf.test`. These tests depend on the choice of an interval of distance values (the argument `rinterval`). A *progress plot* or *envelope representation* of the test (Baddeley et al, 2014) is a plot of the test statistic (and the corresponding critical value) against the length of the interval `rinterval`.

The command dclf.progress performs dclf.test on X using all possible intervals of the form $[0, R]$, and returns the resulting values of the test statistic, and the corresponding critical values of the test, as a function of R.

Similarly mad.progress performs mad.test using all possible intervals and returns the test statistic and critical value.

More generally, mctest.progress performs a test based on the L^p discrepancy between the curves. The deviation between two curves is measured by the pth root of the integral of the pth power of the absolute value of the difference between the two curves. The exponent p is given by the argument exponent. The case exponent=2 is the Cressie-Loosmore-Ford test, while exponent=Inf is the MAD test.

If the argument rmin is given, it specifies the left endpoint of the interval defining the test statistic: the tests are performed using intervals $[r_{\min}, R]$ where $R \geq r_{\min}$.

The result of each command is an object of class "fv" that can be plotted to obtain the progress plot. The display shows the test statistic (solid black line) and the Monte Carlo acceptance region (grey shading).

The significance level for the Monte Carlo test is nrank/(nsim+1). Note that nsim defaults to 99, so if the values of nrank and nsim are not given, the default is a test with significance level 0.01.

If X is an envelope object, then some of the data stored in X may be re-used:

- If X is an envelope object containing simulated functions, and fun=NULL, then the code will re-use the simulated functions stored in X.
- If X is an envelope object containing simulated point patterns, then fun will be applied to the stored point patterns to obtain the simulated functions. If fun is not specified, it defaults to Lest.
- Otherwise, new simulations will be performed, and fun defaults to Lest.

Value

An object of class "fv" that can be plotted to obtain the progress plot.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Andrew Hardegen, Tom Lawrence, Gopal Nair and Robin Milne.

References

Baddeley, A., Diggle, P., Hardegen, A., Lawrence, T., Milne, R. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84** (3) 477–489.

See Also

dclf.test and mad.test for the tests.

See plot.fv for information on plotting objects of class "fv".

Examples

```
plot(dclf.progress(cells, nsim=19))
```

dclf.sigtrace — *Significance Trace of Cressie-Loosmore-Ford or Maximum Absolute Deviation Test*

Description

Generates a Significance Trace of the Diggle(1986)/ Cressie (1991)/ Loosmore and Ford (2006) test or the Maximum Absolute Deviation test for a spatial point pattern.

Usage

```
dclf.sigtrace(X, ...)
mad.sigtrace(X, ...)
mctest.sigtrace(X, fun=Lest, ...,
                exponent=1, interpolate=FALSE, alpha=0.05,
                confint=TRUE, rmin=0)
```

Arguments

`X`	Either a point pattern (object of class "ppp", "lpp" or other class), a fitted point process model (object of class "ppm", "kppm" or other class) or an envelope object (class "envelope").
`...`	Arguments passed to `envelope` or `mctest.progress`. Useful arguments include `fun` to determine the summary function, `nsim` to specify the number of Monte Carlo simulations, `alternative` to specify a one-sided test, and `verbose=FALSE` to turn off the messages.
`fun`	Function that computes the desired summary statistic for a point pattern.
`exponent`	Positive number. The exponent of the L^p distance. See Details.
`interpolate`	Logical value specifying whether to calculate the p-value by interpolation. If `interpolate=FALSE` (the default), a standard Monte Carlo test is performed, yielding a p-value of the form $(k+1)/(n+1)$ where n is the number of simulations and k is the number of simulated values which are more extreme than the observed value. If `interpolate=TRUE`, the p-value is calculated by applying kernel density estimation to the simulated values, and computing the tail probability for this estimated distribution.
`alpha`	Significance level to be plotted (this has no effect on the calculation but is simply plotted as a reference value).
`confint`	Logical value indicating whether to compute a confidence interval for the 'true' p-value.
`rmin`	Optional. Left endpoint for the interval of r values on which the test statistic is calculated.

Details

The Diggle (1986)/ Cressie (1991)/Loosmore and Ford (2006) test and the Maximum Absolute Deviation test for a spatial point pattern are described in `dclf.test`. These tests depend on the choice of an interval of distance values (the argument `rinterval`). A *significance trace* (Bowman and Azzalini, 1997; Baddeley et al, 2014, 2015) of the test is a plot of the p-value obtained from the test against the length of the interval `rinterval`.

The command `dclf.sigtrace` performs `dclf.test` on `X` using all possible intervals of the form $[0, R]$, and returns the resulting p-values as a function of R.

Similarly `mad.sigtrace` performs `mad.test` using all possible intervals and returns the p-values.

More generally, `mctest.sigtrace` performs a test based on the L^p discrepancy between the curves. The deviation between two curves is measured by the pth root of the integral of the pth power of the absolute value of the difference between the two curves. The exponent p is given by the argument `exponent`. The case `exponent=2` is the Cressie-Loosmore-Ford test, while `exponent=Inf` is the MAD test.

If the argument `rmin` is given, it specifies the left endpoint of the interval defining the test statistic: the tests are performed using intervals $[r_{\min}, R]$ where $R \geq r_{\min}$.

The result of each command is an object of class `"fv"` that can be plotted to obtain the significance trace. The plot shows the Monte Carlo p-value (solid black line), the critical value `0.05` (dashed red line), and a pointwise 95% confidence band (grey shading) for the 'true' (Neyman-Pearson) p-value. The confidence band is based on the Agresti-Coull (1998) confidence interval for a binomial proportion (when `interpolate=FALSE`) or the delta method and normal approximation (when `interpolate=TRUE`).

If `X` is an envelope object and `fun=NULL` then the code will re-use the simulated functions stored in `X`.

Value

An object of class `"fv"` that can be plotted to obtain the significance trace.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Agresti, A. and Coull, B.A. (1998) Approximate is better than "Exact" for interval estimation of binomial proportions. *American Statistician* **52**, 119–126.

Baddeley, A., Diggle, P., Hardegen, A., Lawrence, T., Milne, R. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84**(3) 477–489.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2015) Pushing the envelope: extensions of graphical Monte Carlo tests. Submitted for publication.

Bowman, A.W. and Azzalini, A. (1997) *Applied smoothing techniques for data analysis: the kernel approach with S-Plus illustrations*. Oxford University Press, Oxford.

See Also

`dclf.test` for the tests; `dclf.progress` for progress plots.

See `plot.fv` for information on plotting objects of class `"fv"`.

See also `dg.sigtrace`.

Examples

```
plot(dclf.sigtrace(cells, Lest, nsim=19))
```

dclf.test *Diggle-Cressie-Loosmore-Ford and Maximum Absolute Deviation Tests*

Description

Perform the Diggle (1986) / Cressie (1991) / Loosmore and Ford (2006) test or the Maximum Absolute Deviation test for a spatial point pattern.

Usage

```
dclf.test(X, ..., alternative=c("two.sided", "less", "greater"),
                  rinterval = NULL, leaveout=1,
                  scale=NULL, clamp=FALSE, interpolate=FALSE)

mad.test(X, ...,  alternative=c("two.sided", "less", "greater"),
                  rinterval = NULL, leaveout=1,
                  scale=NULL, clamp=FALSE, interpolate=FALSE)
```

Arguments

- `X` — Data for the test. Either a point pattern (object of class "ppp", "lpp" or other class), a fitted point process model (object of class "ppm", "kppm" or other class), a simulation envelope (object of class "envelope") or a previous result of `dclf.test` or `mad.test`.
- `...` — Arguments passed to `envelope`. Useful arguments include `fun` to determine the summary function, `nsim` to specify the number of Monte Carlo simulations, `verbose=FALSE` to turn off the messages, `savefuns` or `savepatterns` to save the simulation results, and `use.theory` described under Details.
- `alternative` — The alternative hypothesis. A character string. The default is a two-sided alternative. See Details.
- `rinterval` — Interval of values of the summary function argument `r` over which the maximum absolute deviation, or the integral, will be computed for the test. A numeric vector of length 2.
- `leaveout` — Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.
- `scale` — Optional. A function in the R language which determines the relative scale of deviations, as a function of distance r. Summary function values for distance `r` will be *divided* by `scale(r)` before the test statistic is computed.
- `clamp` — Logical value indicating how to compute deviations in a one-sided test. Deviations of the observed summary function from the theoretical summary function are initially evaluated as signed real numbers, with large positive values indicating consistency with the alternative hypothesis. If `clamp=FALSE` (the default), these values are not changed. If `clamp=TRUE`, any negative values are replaced by zero.
- `interpolate` — Logical value specifying whether to calculate the p-value by interpolation. If `interpolate=FALSE` (the default), a standard Monte Carlo test is performed,

yielding a p-value of the form $(k+1)/(n+1)$ where n is the number of simulations and k is the number of simulated values which are more extreme than the observed value. If `interpolate=TRUE`, the p-value is calculated by applying kernel density estimation to the simulated values, and computing the tail probability for this estimated distribution.

Details

These functions perform hypothesis tests for goodness-of-fit of a point pattern dataset to a point process model, based on Monte Carlo simulation from the model.

`dclf.test` performs the test advocated by Loosmore and Ford (2006) which is also described in Diggle (1986), Cressie (1991, page 667, equation (8.5.42)) and Diggle (2003, page 14). See Baddeley et al (2014) for detailed discussion.

`mad.test` performs the 'global' or 'Maximum Absolute Deviation' test described by Ripley (1977, 1981). See Baddeley et al (2014).

The type of test depends on the type of argument `X`.

- If `X` is some kind of point pattern, then a test of Complete Spatial Randomness (CSR) will be performed. That is, the null hypothesis is that the point pattern is completely random.
- If `X` is a fitted point process model, then a test of goodness-of-fit for the fitted model will be performed. The model object contains the data point pattern to which it was originally fitted. The null hypothesis is that the data point pattern is a realisation of the model.
- If `X` is an envelope object generated by `envelope`, then it should have been generated with `savefuns=TRUE` or `savepatterns=TRUE` so that it contains simulation results. These simulations will be treated as realisations from the null hypothesis.
- Alternatively `X` could be a previously-performed test of the same kind (i.e. the result of calling `dclf.test` or `mad.test`). The simulations used to perform the original test will be re-used to perform the new test (provided these simulations were saved in the original test, by setting `savefuns=TRUE` or `savepatterns=TRUE`).

The argument `alternative` specifies the alternative hypothesis, that is, the direction of deviation that will be considered statistically significant. If `alternative="two.sided"` (the default), both positive and negative deviations (between the observed summary function and the theoretical function) are significant. If `alternative="less"`, then only negative deviations (where the observed summary function is lower than the theoretical function) are considered. If `alternative="greater"`, then only positive deviations (where the observed summary function is higher than the theoretical function) are considered.

In all cases, the algorithm will first call `envelope` to generate or extract the simulated summary functions. The number of simulations that will be generated or extracted, is determined by the argument `nsim`, and defaults to 99. The summary function that will be computed is determined by the argument `fun` (or the first unnamed argument in the list `...`) and defaults to `Kest` (except when `X` is an envelope object generated with `savefuns=TRUE`, when these functions will be taken).

The choice of summary function `fun` affects the power of the test. It is normally recommended to apply a variance-stabilising transformation (Ripley, 1981). If you are using the K function, the normal practice is to replace this by the L function (Besag, 1977) computed by `Lest`. If you are using the F or G functions, the recommended practice is to apply Fisher's variance-stabilising transformation $\sin^{-1}\sqrt{x}$ using the argument `transform`. See the Examples.

The argument `rinterval` specifies the interval of distance values r which will contribute to the test statistic (either maximising over this range of values for `mad.test`, or integrating over this range of values for `dclf.test`). This affects the power of the test. General advice and experiments in Baddeley et al (2014) suggest that the maximum r value should be slightly larger than the maximum possible range of interaction between points. The `dclf.test` is quite sensitive to this choice, while the `mad.test` is relatively insensitive.

It is also possible to specify a pointwise test (i.e. taking a single, fixed value of distance r) by specifing `rinterval = c(r,r)`.

The argument `use.theory` passed to `envelope` determines whether to compare the summary function for the data to its theoretical value for CSR (`use.theory=TRUE`) or to the sample mean of simulations from CSR (`use.theory=FALSE`).

The argument `leaveout` specifies how to calculate the discrepancy between the summary function for the data and the nominal reference value, when the reference value must be estimated by simulation. The values `leaveout=0` and `leaveout=1` are both algebraically equivalent (Baddeley et al, 2014, Appendix) to computing the difference `observed - reference` where the `reference` is the mean of simulated values. The value `leaveout=2` gives the leave-two-out discrepancy proposed by Dao and Genton (2014).

Value

An object of class `"htest"`. Printing this object gives a report on the result of the test. The p-value is contained in the component `p.value`.

Handling Ties

If the observed value of the test statistic is equal to one or more of the simulated values (called a *tied value*), then the tied values will be assigned a random ordering, and a message will be printed.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Andrew Hardegen and Suman Rakshit.

References

Baddeley, A., Diggle, P.J., Hardegen, A., Lawrence, T., Milne, R.K. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84**(3) 477–489.

Baddeley, A., Hardegen, A., Lawrence, T., Milne, R.K. and Nair, G. (2015) *Pushing the envelope*. In preparation.

Besag, J. (1977) Discussion of Dr Ripley's paper. *Journal of the Royal Statistical Society, Series B*, **39**, 193–195.

Cressie, N.A.C. (1991) *Statistics for spatial data*. John Wiley and Sons, 1991.

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neuroscience Methods* **18**, 115–125.

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

Loosmore, N.B. and Ford, E.D. (2006) Statistical inference using the *G* or *K* point pattern spatial statistics. *Ecology* **87**, 1925–1931.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

See Also

envelope, dclf.progress

Examples

```
dclf.test(cells, Lest, nsim=39)
m <- mad.test(cells, Lest, verbose=FALSE, rinterval=c(0, 0.1), nsim=19)
m
# extract the p-value
m$p.value
# variance stabilised G function
dclf.test(cells, Gest, transform=expression(asin(sqrt(.))),
                 verbose=FALSE, nsim=19)

## one-sided test
ml <- mad.test(cells, Lest, verbose=FALSE, nsim=19, alternative="less")

## scaled
mad.test(cells, Kest, verbose=FALSE, nsim=19,
         rinterval=c(0.05, 0.2),
         scale=function(r) { r })
```

---

default.dummy — *Generate a Default Pattern of Dummy Points*

Description

Generates a default pattern of dummy points for use in a quadrature scheme.

Usage

```
default.dummy(X, nd, random=FALSE, ntile=NULL, npix=NULL,
              quasi=FALSE, ..., eps=NULL, verbose=FALSE)
```

Arguments

X	The observed data point pattern. An object of class "ppp" or in a format recognised by as.ppp()
nd	Optional. Integer, or integer vector of length 2, specifying an nd * nd or nd[1] * nd[2] rectangular array of dummy points.
random	Logical value. If TRUE, the dummy points are generated randomly.
quasi	Logical value. If TRUE, the dummy points are generated by a quasirandom sequence.
ntile	Optional. Integer or pair of integers specifying the number of rows and columns of tiles used in the counting rule.
npix	Optional. Integer or pair of integers specifying the number of rows and columns of pixels used in computing approximate areas.
...	Ignored.
eps	Optional. Grid spacing. A positive number, or a vector of two positive numbers, giving the horizontal and vertical spacing, respectively, of the grid of dummy points. Incompatible with nd.
verbose	If TRUE, information about the construction of the quadrature scheme is printed.

Details

This function provides a sensible default for the dummy points in a quadrature scheme.

A quadrature scheme consists of the original data point pattern, an additional pattern of dummy points, and a vector of quadrature weights for all these points. See `quad.object` for further information about quadrature schemes.

If `random` and `quasi` are both false (the default), then the function creates dummy points in a regular `nd[1]` by `nd[1]` rectangular grid. If `random` is true and `quasi` is false, then the frame of the window is divided into an `nd[1]` by `nd[1]` array of tiles, and one dummy point is generated at random inside each tile. If `quasi` is true, a quasirandom pattern of `nd[1] * nd[2]` points is generated. In all cases, the four corner points of the frame of the window are added. Then if the window is not rectangular, any dummy points lying outside it are deleted.

If `nd` is missing, a default value (depending on the data pattern `X`) is computed by `default.ngrid`.

Alternative functions for creating dummy patterns include `corners`, `gridcentres`, `stratrand` and `spokes`.

Value

A point pattern (an object of class `"ppp"`, see `ppp.object`) containing the dummy points.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`quad.object`, `quadscheme`, `corners`, `gridcentres`, `stratrand`, `spokes`

Examples

```
data(simdat)
P <- simdat
D <- default.dummy(P, 100)
## Not run: plot(D)
Q <- quadscheme(P, D, "grid")
## Not run: plot(union.quad(Q))
```

`default.expand` *Default Expansion Rule for Simulation of Model*

Description

Defines the default expansion window or expansion rule for simulation of a fitted point process model.

Usage

```
default.expand(object, m=2, epsilon=1e-6, w=Window(object))
```

Arguments

object — A point process model (object of class "ppm" or "rmhmodel").

m — A single numeric value. The window will be expanded by a distance `m * reach(object)` along each side.

epsilon — Threshold argument passed to `reach` to determine `reach(object)`.

w — Optional. The un-expanded window in which the model is defined. The resulting simulated point patterns will lie in this window.

Details

This function computes a default value for the expansion rule (the argument `expand` in `rmhcontrol`) given a fitted point process model `object`. This default is used by `envelope`, `qqplot.ppm`, `simulate.ppm` and other functions.

Suppose we wish to generate simulated realisations of a fitted point process model inside a window `w`. It is advisable to first simulate the pattern on a larger window, and then clip it to the original window `w`. This avoids edge effects in the simulation. It is called *expansion* of the simulation window.

Accordingly, for the Metropolis-Hastings simulation algorithm `rmh`, the algorithm control parameters specified by `rmhcontrol` include an argument `expand` that determines the expansion of the simulation window.

The function `default.expand` determines the default expansion rule for a fitted point process model `object`.

If the model is Poisson, then no expansion is necessary. No expansion is performed by default, and `default.expand` returns a rule representing no expansion. The simulation window is the original window `w = Window(object)`.

If the model depends on external covariates (i.e.\ covariates other than the Cartesian covariates `x` and `y` and the `marks`) then no expansion is feasible, in general, because the spatial domain of the covariates is not guaranteed to be large enough. `default.expand` returns a rule representing no expansion. The simulation window is the original window `w = Window(object)`.

If the model depends on the Cartesian covariates `x` and `y`, it would be feasible to expand the simulation window, and this was the default for **spatstat** version 1.24-1 and earlier. However this sometimes produces artefacts (such as an empty point pattern) or memory overflow, because the fitted trend, extrapolated outside the original window of the data, may become very large. In **spatstat** version 1.24-2 and later, the default rule is *not* to expand if the model depends on `x` or `y`. Again `default.expand` returns a rule representing no expansion.

Otherwise, expansion will occur. The original window `w = Window(object)` is expanded by a distance `m * rr`, where `rr` is the interaction range of the model, computed by `reach`. If `w` is a rectangle then each edge of `w` is displaced outward by distance `m * rr`. If `w` is not a rectangle then `w` is dilated by distance `m * rr` using `dilation`.

Value

A window expansion rule (object of class "rmhexpand").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rmhexpand, rmhcontrol, rmh, envelope, qqplot.ppm

Examples

```
data(cells)
fit <- ppm(cells, ~1, Strauss(0.07))
default.expand(fit)
mod <- rmhmodel(cif="strauss", par=list(beta=100, gamma=0.5, r=0.07))
default.expand(fit)
```

`default.rmhcontrol` *Set Default Control Parameters for Metropolis-Hastings Algorithm.*

Description

Given a fitted point process model, this command sets appropriate default values of the parameters controlling the iterative behaviour of the Metropolis-Hastings algorithm.

Usage

```
default.rmhcontrol(model, w=NULL)
```

Arguments

model	A fitted point process model (object of class "ppm")
w	Optional. Window for the resulting simulated patterns.

Details

This function sets the values of the parameters controlling the iterative behaviour of the Metropolis-Hastings simulation algorithm. It uses default values that would be appropriate for the fitted point process model `model`.

The expansion parameter expand is set to `default.expand(model, w)`.

All other parameters revert to their defaults given in `rmhcontrol.default`.

See `rmhcontrol` for the full list of control parameters. To override default parameters, use `update.rmhcontrol`.

Value

An object of class "rmhcontrol". See `rmhcontrol`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rmhcontrol, update.rmhcontrol, ppm, default.expand

Examples

```
fit <- ppm(cells, ~1, Strauss(0.1))
default.rmhcontrol(fit)
default.rmhcontrol(fit, w=square(2))
```

---

delaunay *Delaunay Triangulation of Point Pattern*

Description

Computes the Delaunay triangulation of a spatial point pattern.

Usage

```
delaunay(X)
```

Arguments

X	Spatial point pattern (object of class "ppp").

Details

The Delaunay triangulation of a spatial point pattern X is defined as follows. First the Dirichlet/Voronoi tessellation of X computed; see dirichlet. Then two points of X are defined to be Delaunay neighbours if their Dirichlet/Voronoi tiles share a common boundary. Every pair of Delaunay neighbours is joined by a straight line. The result is a tessellation, consisting of disjoint triangles. The union of these triangles is the convex hull of X.

Value

A tessellation (object of class "tess"). The window of the tessellation is the convex hull of X, not the original window of X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

tess, dirichlet, convexhull.xy, ppp, delaunayDistance, delaunayNetwork

Examples

```
X <- runifpoint(42)
plot(delaunay(X))
plot(X, add=TRUE)
```

`delaunayDistance` *Distance on Delaunay Triangulation*

Description

Computes the graph distance in the Delaunay triangulation of a point pattern.

Usage

```
delaunayDistance(X)
```

Arguments

`X`	Spatial point pattern (object of class "ppp").

Details

The Delaunay triangulation of a spatial point pattern `X` is defined as follows. First the Dirichlet/Voronoi tessellation of `X` computed; see `dirichlet`. Then two points of `X` are defined to be Delaunay neighbours if their Dirichlet/Voronoi tiles share a common boundary. Every pair of Delaunay neighbours is joined by a straight line.

The *graph distance* in the Delaunay triangulation between two points `X[i]` and `X[j]` is the minimum number of edges of the Delaunay triangulation that must be traversed to go from `X[i]` to `X[j]`.

This command returns a matrix `D` such that `D[i,j]` is the graph distance between `X[i]` and `X[j]`.

Value

A symmetric square matrix with integer entries.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`delaunay`, `delaunayNetwork`

Examples

```
X <- runifpoint(20)
M <- delaunayDistance(X)
plot(delaunay(X), lty=3)
text(X, labels=M[1, ], cex=2)
```

delaunayNetwork *Linear Network of Delaunay Triangulation or Dirichlet Tessellation*

Description

Computes the edges of the Delaunay triangulation or Dirichlet tessellation of a point pattern, and returns the result as a linear network object.

Usage

```
delaunayNetwork(X)

dirichletNetwork(X, ...)
```

Arguments

`X`	A point pattern (object of class "ppp").
`...`	Arguments passed to `as.linnet.psp`

Details

For `delaunayNetwork`, points of `X` which are neighbours in the Delaunay triangulation (see `delaunay`) will be joined by a straight line. The result will be returned as a linear network (object of class "linnet").

For `dirichletNetwork`, the Dirichlet tessellation is computed (see `dirichlet`) and the edges of the tiles of the tessellation are extracted. This is converted to a linear network using `as.linnet.psp`.

Value

Linear network (object of class "linnet") or `NULL`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`delaunay`, `dirichlet`, `delaunayDistance`

Examples

```
LE <- delaunayNetwork(cells)
LI <- dirichletNetwork(cells)
```

deletebranch *Delete or Extract a Branch of a Tree*

Description

Deletes or extracts a given branch of a tree.

Usage

```
deletebranch(X, ...)

## S3 method for class 'linnet'
deletebranch(X, code, labels, ...)

## S3 method for class 'lpp'
deletebranch(X, code, labels, ...)

extractbranch(X, ...)

## S3 method for class 'linnet'
extractbranch(X, code, labels, ..., which=NULL)

## S3 method for class 'lpp'
extractbranch(X, code, labels, ..., which=NULL)
```

Arguments

`X`	Linear network (object of class `"linnet"`) or point pattern on a linear network (object of class `"lpp"`).
`code`	Character string. Label of the branch to be deleted or extracted.
`labels`	Vector of character strings. Branch labels for the vertices of the network, usually obtained from `treebranchlabels`.
`...`	Arguments passed to methods.
`which`	Logical vector indicating which vertices of the network should be extracted. Overrides `code` and `labels`.

Details

The linear network `L <- X` or `L <- as.linnet(X)` must be a tree, that is, it has no loops.

The argument `labels` should be a character vector giving tree branch labels for each vertex of the network. It is usually obtained by calling `treebranchlabels`.

The branch designated by the string `code` will be deleted or extracted.

The return value is the result of deleting or extracting this branch from `X` along with any data associated with this branch (such as points or marks).

Value

Another object of the same type as `X` obtained by deleting or extracting the specified branch.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

treebranchlabels, branchlabelfun, linnet

Examples

```
# make a simple tree
m <- simplenet$m
m[8,10] <- m[10,8] <- FALSE
L <- linnet(vertices(simplenet), m)
plot(L, main="")
# compute branch labels
tb <- treebranchlabels(L, 1)
tbc <- paste0("[", tb, "]")
text(vertices(L), labels=tbc, cex=2)

# delete branch B
LminusB <- deletebranch(L, "b", tb)
plot(LminusB, add=TRUE, col="green")

# extract branch B
LB <- extractbranch(L, "b", tb)
plot(LB, add=TRUE, col="red")
```

---

deltametric *Delta Metric*

Description

Computes the discrepancy between two sets A and B according to Baddeley's delta-metric.

Usage

```
deltametric(A, B, p = 2, c = Inf, ...)
```

Arguments

A,B	The two sets which will be compared. Windows (objects of class "owin"), point patterns (objects of class "ppp") or line segment patterns (objects of class "psp").
p	Index of the L^p metric. Either a positive numeric value, or Inf.
c	Distance threshold. Either a positive numeric value, or Inf.
...	Arguments passed to as.mask to determine the pixel resolution of the distance maps computed by distmap.

Details

Baddeley (1992a, 1992b) defined a distance between two sets A and B contained in a space W by

$$\Delta(A,B) = \left[\frac{1}{|W|} \int_W |\min(c, d(x,A)) - \min(c, d(x,B))|^p \, \mathrm{d}x \right]^{1/p}$$

where $c \geq 0$ is a distance threshold parameter, $0 < p \leq \infty$ is the exponent parameter, and $d(x,A)$ denotes the shortest distance from a point x to the set A. Also `|W|` denotes the area or volume of the containing space W.

This is defined so that it is a *metric*, i.e.

- $\Delta(A,B) = 0$ if and only if $A = B$
- $\Delta(A,B) = \Delta(B,A)$
- $\Delta(A,C) \leq \Delta(A,B) + \Delta(B,C)$

It is topologically equivalent to the Hausdorff metric (Baddeley, 1992a) but has better stability properties in practical applications (Baddeley, 1992b).

If $p = \infty$ and $c = \infty$ the Delta metric is equal to the Hausdorff metric.

The algorithm uses `distmap` to compute the distance maps $d(x,A)$ and $d(x,B)$, then approximates the integral numerically. The accuracy of the computation depends on the pixel resolution which is controlled through the extra arguments `...` passed to `as.mask`.

Value

A numeric value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A.J. (1992a) Errors in binary images and an L^p version of the Hausdorff metric. *Nieuw Archief voor Wiskunde* **10**, 157–183.

Baddeley, A.J. (1992b) An error metric for binary images. In W. Foerstner and S. Ruwiedel (eds) *Robust Computer Vision*. Karlsruhe: Wichmann. Pages 59–78.

See Also

`distmap`

Examples

```
X <- runifpoint(20)
Y <- runifpoint(10)
deltametric(X, Y, p=1,c=0.1)
```

density.lpp *Kernel Estimate of Intensity on a Linear Network*

Description

Estimates the intensity of a point process on a linear network by applying kernel smoothing to the point pattern data.

Usage

```
## S3 method for class 'lpp'
density(x, sigma, ...,
        weights=NULL,
        kernel="gaussian",
        continuous=TRUE,
        epsilon = 1e-06, verbose = TRUE,
        debug = FALSE, savehistory = TRUE,
        old=FALSE)

## S3 method for class 'splitppx'
density(x, sigma, ...)
```

Arguments

`x`	Point pattern on a linear network (object of class `"lpp"`) to be smoothed.
`sigma`	Smoothing bandwidth (standard deviation of the kernel) in the same units as the spatial coordinates of `x`.
`...`	Arguments passed to `as.mask` determining the resolution of the result.
`weights`	Optional. Numeric vector of weights associated with the points of `x`. Weights may be positive, negative or zero.
`kernel`	Character string specifying the smoothing kernel. See `dkernel` for possible options.
`continuous`	Logical value indicating whether to compute the "equal-split continuous" smoother (`continuous=TRUE`, the default) or the "equal-split discontinuous" smoother (`continuous=FALSE`).
`epsilon`	Tolerance value. A tail of the kernel with total mass less than `epsilon` may be deleted.
`verbose`	Logical value indicating whether to print progress reports.
`debug`	Logical value indicating whether to print debugging information.
`savehistory`	Logical value indicating whether to save the entire history of the algorithm, for the purposes of evaluating performance.
`old`	Logical value indicating whether to use the old, very slow algorithm for the equal-split continuous estimator.

Details

Kernel smoothing is applied to the points of x using one of the rules described in Okabe and Sugihara (2012) and McSwiggan et al (2016). The result is a pixel image on the linear network (class "linim") which can be plotted.

If continuous=TRUE (the default), smoothing is performed using the "equal-split continuous" rule described in Section 9.2.3 of Okabe and Sugihara (2012). The resulting function is continuous on the linear network.

If continuous=FALSE, smoothing is performed using the "equal-split discontinuous" rule described in Section 9.2.2 of Okabe and Sugihara (2012). The resulting function is not continuous.

In the default case (where continuous=TRUE and kernel="gaussian" and old=FALSE), computation is performed rapidly by solving the classical heat equation on the network, as described in McSwiggan et al (2016). Computational time is short, but increases quadratically with sigma. The arguments epsilon,debug,verbose,savehistory are ignored.

In all other cases, computation is performed by path-tracing as described in Okabe and Sugihara (2012); computation can be extremely slow, and time increases exponentially with sigma.

There is also a method for split point patterns on a linear network (class "splitppx") which will return a list of pixel images.

Value

A pixel image on the linear network (object of class "linim").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Greg McSwiggan.

References

McSwiggan, G., Baddeley, A. and Nair, G. (2016) Kernel density estimation on a linear network. *Scandinavian Journal of Statistics*, In press.

Okabe, A. and Sugihara, K. (2012) *Spatial analysis along networks*. Wiley.

See Also

lpp, linim

Examples

```
X <- runiflpp(3, simplenet)
D <- density(X, 0.2, verbose=FALSE)
plot(D, style="w", main="", adjust=2)
Dw <- density(X, 0.2, weights=c(1,2,-1), verbose=FALSE)
De <- density(X, 0.2, kernel="epanechnikov", verbose=FALSE)
Ded <- density(X, 0.2, kernel="epanechnikov", continuous=FALSE, verbose=FALSE)
```

density.ppp *Kernel Smoothed Intensity of Point Pattern*

Description

Compute a kernel smoothed intensity function from a point pattern.

Usage

```
## S3 method for class 'ppp'
density(x, sigma=NULL, ...,
        weights=NULL, edge=TRUE, varcov=NULL,
        at="pixels", leaveoneout=TRUE,
        adjust=1, diggle=FALSE, se=FALSE,
        kernel="gaussian",
        scalekernel=is.character(kernel),
        positive=FALSE, verbose=TRUE)
```

Arguments

x	Point pattern (object of class "ppp").
sigma	Standard deviation of isotropic smoothing kernel. Either a numerical value, or a function that computes an appropriate value of sigma.
weights	Optional weights to be attached to the points. A numeric vector, numeric matrix, an expression, or a pixel image.
...	Additional arguments passed to pixellate.ppp and as.mask to determine the pixel resolution, or passed to sigma if it is a function.
edge	Logical value indicating whether to apply edge correction.
varcov	Variance-covariance matrix of anisotropic smoothing kernel. Incompatible with sigma.
at	String specifying whether to compute the intensity values at a grid of pixel locations (at="pixels") or only at the points of x (at="points").
leaveoneout	Logical value indicating whether to compute a leave-one-out estimator. Applicable only when at="points".
adjust	Optional. Adjustment factor for the smoothing parameter.
diggle	Logical. If TRUE, use the Jones-Diggle improved edge correction, which is more accurate but slower to compute than the default correction.
kernel	The smoothing kernel. A character string specifying the smoothing kernel (current options are "gaussian", "epanechnikov", "quartic" or "disc"), or a pixel image (object of class "im") containing values of the kernel, or a function(x,y) which yields values of the kernel.
scalekernel	Logical value. If scalekernel=TRUE, then the kernel will be rescaled to the bandwidth determined by sigma and varcov: this is the default behaviour when kernel is a character string. If scalekernel=FALSE, then sigma and varcov will be ignored: this is the default behaviour when kernel is a function or a pixel image.
se	Logical value indicating whether to compute standard errors as well.

positive
: Logical value indicating whether to force all density values to be positive numbers. Default is FALSE.

verbose
: Logical value indicating whether to issue warnings about numerical problems and conditions.

Details

This is a method for the generic function `density`.

It computes a fixed-bandwidth kernel estimate (Diggle, 1985) of the intensity function of the point process that generated the point pattern `x`.

By default it computes the convolution of the isotropic Gaussian kernel of standard deviation `sigma` with point masses at each of the data points in `x`. Anisotropic Gaussian kernels are also supported. Each point has unit weight, unless the argument `weights` is given.

If `edge=TRUE`, the intensity estimate is corrected for edge effect bias in one of two ways:

- If `diggle=FALSE` (the default) the intensity estimate is correted by dividing it by the convolution of the Gaussian kernel with the window of observation. This is the approach originally described in Diggle (1985). Thus the intensity value at a point u is

$$\hat{\lambda}(u) = e(u) \sum_i k(x_i - u) w_i$$

  where k is the Gaussian smoothing kernel, $e(u)$ is an edge correction factor, and w_i are the weights.
- If `diggle=TRUE` then the code uses the improved edge correction described by Jones (1993) and Diggle (2010, equation 18.9). This has been shown to have better performance (Jones, 1993) but is slightly slower to compute. The intensity value at a point u is

$$\hat{\lambda}(u) = \sum_i k(x_i - u) w_i e(x_i)$$

  where again k is the Gaussian smoothing kernel, $e(x_i)$ is an edge correction factor, and w_i are the weights.

In both cases, the edge correction term $e(u)$ is the reciprocal of the kernel mass inside the window:

$$\frac{1}{e(u)} = \int_W k(v - u) \, \mathrm{d}v$$

where W is the observation window.

The smoothing kernel is determined by the arguments `sigma`, `varcov` and `adjust`.

- if `sigma` is a single numerical value, this is taken as the standard deviation of the isotropic Gaussian kernel.
- alternatively `sigma` may be a function that computes an appropriate bandwidth for the isotropic Gaussian kernel from the data point pattern by calling `sigma(x)`. To perform automatic bandwidth selection using cross-validation, it is recommended to use the functions `bw.diggle` or `bw.ppl`.
- The smoothing kernel may be chosen to be any Gaussian kernel, by giving the variance-covariance matrix `varcov`. The arguments `sigma` and `varcov` are incompatible.
- Alternatively `sigma` may be a vector of length 2 giving the standard deviations of two independent Gaussian coordinates, thus equivalent to `varcov = diag(rep(sigma^2, 2))`.

- if neither sigma nor varcov is specified, an isotropic Gaussian kernel will be used, with a default value of sigma calculated by a simple rule of thumb that depends only on the size of the window.
- The argument adjust makes it easy for the user to change the bandwidth specified by any of the rules above. The value of sigma will be multiplied by the factor adjust. The matrix varcov will be multiplied by adjust^2. To double the smoothing bandwidth, set adjust=2.

If at="pixels" (the default), intensity values are computed at every location u in a fine grid, and are returned as a pixel image. The point pattern is first discretised using pixellate.ppp, then the intensity is computed using the Fast Fourier Transform. Accuracy depends on the pixel resolution and the discretisation rule. The pixel resolution is controlled by the arguments ... passed to as.mask (specify the number of pixels by dimyx or the pixel size by eps). The discretisation rule is controlled by the arguments ... passed to pixellate.ppp (the default rule is that each point is allocated to the nearest pixel centre; this can be modified using the arguments fractional and preserve).

If at="points", the intensity values are computed to high accuracy at the points of x only. Computation is performed by directly evaluating and summing the Gaussian kernel contributions without discretising the data. The result is a numeric vector giving the density values. The intensity value at a point x_i is (if diggle=FALSE)

$$\hat{\lambda}(x_i) = e(x_i) \sum_j k(x_j - x_i) w_j$$

or (if diggle=TRUE)

$$\hat{\lambda}(x_i) = \sum_j k(x_j - x_i) w_j e(x_j)$$

If leaveoneout=TRUE (the default), then the sum in the equation is taken over all j not equal to i, so that the intensity value at a data point is the sum of kernel contributions from all *other* data points. If leaveoneout=FALSE then the sum is taken over all j, so that the intensity value at a data point includes a contribution from the same point.

If weights is a matrix with more than one column, then the calculation is effectively repeated for each column of weights. The result is a list of images (if at="pixels") or a matrix of numerical values (if at="points").

The argument weights can also be an expression. It will be evaluated in the data frame as.data.frame(x) to obtain a vector or matrix of weights. The expression may involve the symbols x and y representing the Cartesian coordinates, the symbol marks representing the mark values if there is only one column of marks, and the names of the columns of marks if there are several columns.

The argument weights can also be a pixel image (object of class "im"). numerical weights for the data points will be extracted from this image (by looking up the pixel values at the locations of the data points in x).

To select the bandwidth sigma automatically by cross-validation, use bw.diggle or bw.ppl.

To perform spatial interpolation of values that were observed at the points of a point pattern, use Smooth.ppp.

For adaptive nonparametric estimation, see adaptive.density. For data sharpening, see sharpen.ppp.

To compute a relative risk surface or probability map for two (or more) types of points, use relrisk.

Value

By default, the result is a pixel image (object of class "im"). Pixel values are estimated intensity values, expressed in “points per unit area”.

If at="points", the result is a numeric vector of length equal to the number of points in x. Values are estimated intensity values at the points of x.

In either case, the return value has attributes "sigma" and "varcov" which report the smoothing bandwidth that was used.

If weights is a matrix with more than one column, then the result is a list of images (if at="pixels") or a matrix of numerical values (if at="points").

If se=TRUE, the result is a list with two elements named estimate and SE, each of the format described above.

Negative Values

Negative and zero values of the density estimate are possible when at="pixels" because of numerical errors in finite-precision arithmetic.

By default, density.ppp does not try to repair such errors. This would take more computation time and is not always needed. (Also it would not be appropriate if weights include negative values.)

To ensure that the resulting density values are always positive, set positive=TRUE.

Note

This function is often misunderstood.

The result of density.ppp is not a spatial smoothing of the marks or weights attached to the point pattern. To perform spatial interpolation of values that were observed at the points of a point pattern, use Smooth.ppp.

The result of density.ppp is not a probability density. It is an estimate of the *intensity function* of the point process that generated the point pattern data. Intensity is the expected number of random points per unit area. The units of intensity are "points per unit area". Intensity is usually a function of spatial location, and it is this function which is estimated by density.ppp. The integral of the intensity function over a spatial region gives the expected number of points falling in this region.

Inspecting an estimate of the intensity function is usually the first step in exploring a spatial point pattern dataset. For more explanation, see Baddeley, Rubak and Turner (2015) or Diggle (2003, 2010).

If you have two (or more) types of points, and you want a probability map or relative risk surface (the spatially-varying probability of a given type), use relrisk.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

Diggle, P.J. (1985) A kernel method for smoothing point process data. *Applied Statistics* (Journal of the Royal Statistical Society, Series C) **34** (1985) 138–147.

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

Diggle, P.J. (2010) Nonparametric methods. Chapter 18, pp. 299–316 in A.E. Gelfand, P.J. Diggle, M. Fuentes and P. Guttorp (eds.) *Handbook of Spatial Statistics*, CRC Press, Boca Raton, FL.

Jones, M.C. (1993) Simple boundary corrections for kernel density estimation. *Statistics and Computing* **3**, 135–146.

See Also

bw.diggle, bw.ppl, Smooth.ppp, sharpen.ppp, adaptive.density, relrisk, ppp.object, im.object

Examples

```
if(interactive()) {
  opa <- par(mfrow=c(1,2))
  plot(density(cells, 0.05))
  plot(density(cells, 0.05, diggle=TRUE))
  par(opa)
  v <- diag(c(0.05, 0.07)^2)
  plot(density(cells, varcov=v))
}

  Z <- density(cells, 0.05)
  Z <- density(cells, 0.05, diggle=TRUE)
  Z <- density(cells, 0.05, se=TRUE)
  Z <- density(cells, varcov=diag(c(0.05^2, 0.07^2)))
  Z <- density(cells, 0.05, weights=data.frame(a=1:42,b=42:1))
  Z <- density(cells, 0.05, weights=expression(x))

# automatic bandwidth selection
plot(density(cells, sigma=bw.diggle(cells)))
# equivalent:
plot(density(cells, bw.diggle))
# evaluate intensity at points
density(cells, 0.05, at="points")

plot(density(cells, sigma=0.4, kernel="epanechnikov"))

# relative risk calculation by hand (see relrisk.ppp)
lung <- split(chorley)$lung
larynx <- split(chorley)$larynx
D <- density(lung, sigma=2)
plot(density(larynx, sigma=2, weights=1/D))
```

---

density.psp *Kernel Smoothing of Line Segment Pattern or Linear Network*

Description

Compute a kernel smoothed intensity function from a line segment pattern or a linear network.

Usage

```
  ## S3 method for class 'psp'
density(x, sigma, ..., edge=TRUE,
                   method=c("FFT", "C", "interpreted"),
                   at=NULL)

  ## S3 method for class 'linnet'
density(x, ...)
```

Arguments

x
: Line segment pattern (object of class "psp") or linear network (object of class "linnet") to be smoothed.

sigma
: Standard deviation of isotropic Gaussian smoothing kernel.

...
: Extra arguments, including arguments passed to as.mask to determine the resolution of the resulting image.

edge
: Logical flag indicating whether to apply edge correction.

method
: Character string (partially matched) specifying the method of computation. Option "FFT" is the fastest, while "C" is the most accurate.

at
: Optional. An object specifying the locations where density values should be computed. Either a window (object of class "owin") or a point pattern (object of class "ppp" or "lpp").

Details

These are methods for the generic function density for the classes "psp" (line segment patterns) and "linnet" (linear networks). If x is a linear network, it is first converted to a line segment pattern.

A kernel estimate of the intensity of the line segment pattern is computed. The result is the convolution of the isotropic Gaussian kernel, of standard deviation sigma, with the line segments. The result is computed as follows:

- if method="FFT" (the default), the line segments are discretised using pixellate.psp, then the Fast Fourier Transform is used to calculate the convolution. This method is the fastest, but is slightly less accurate. Accuracy can be improved by increasing pixel resolution.
- if method="C" the exact value of the convolution at the centre of each pixel is computed analytically using C code;
- if method="interpreted", the exact value of the convolution at the centre of each pixel is computed analytically using R code. This method is the slowest.

If edge=TRUE this result is adjusted for edge effects by dividing it by the convolution of the same Gaussian kernel with the observation window.

If the argument at is given, then it specifies the locations where density values should be computed.

- If at is a window, then the window is converted to a binary mask using the arguments ..., and density values are computed at the centre of each pixel in this mask. The result is a pixel image.
- If at is a point pattern, then density values are computed at each point location, and the result is a numeric vector.

Value

A pixel image (object of class "im") or a numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

psp.object, im.object, density

Examples

```
L <- psp(runif(20),runif(20),runif(20),runif(20), window=owin())
D <- density(L, sigma=0.03)
plot(D, main="density(L)")
plot(L, add=TRUE)
```

---

density.splitppp *Kernel Smoothed Intensity of Split Point Pattern*

Description

Compute a kernel smoothed intensity function for each of the components of a split point pattern, or each of the point patterns in a list.

Usage

```
  ## S3 method for class 'splitppp'
density(x, ..., se=FALSE)

  ## S3 method for class 'ppplist'
density(x, ..., se=FALSE)
```

Arguments

x	Split point pattern (object of class "splitppp" created by split.ppp) to be smoothed. Alternatively a list of point patterns, of class "ppplist".
...	Arguments passed to density.ppp to control the smoothing, pixel resolution, edge correction etc.
se	Logical value indicating whether to compute standard errors as well.

Details

This is a method for the generic function density.

The argument x should be a list of point patterns, and should belong to one of the classes "ppplist" or "splitppp".

Typically x is obtained by applying the function split.ppp to a point pattern y by calling split(y). This splits the points of y into several sub-patterns.

A kernel estimate of the intensity function of each of the point patterns is computed using density.ppp.

The return value is usually a list, each of whose entries is a pixel image (object of class "im"). The return value also belongs to the class "solist" and can be plotted or printed.

If the argument at="points" is given, the result is a list of numeric vectors giving the intensity values at the data points.

If se=TRUE, the result is a list with two elements named estimate and SE, each of the format described above.

Value

A list of pixel images (objects of class "im") which can be plotted or printed; or a list of numeric vectors giving the values at specified points.

If se=TRUE, the result is a list with two elements named estimate and SE, each of the format described above.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppp.object, im.object

Examples

```
Z <- density(split(amacrine), 0.05)
plot(Z)
```

---

deriv.fv *Calculate Derivative of Function Values*

Description

Applies numerical differentiation to the values in selected columns of a function value table.

Usage

```
## S3 method for class 'fv'
deriv(expr, which = "*", ...,
          method=c("spline", "numeric"),
          kinks=NULL,
          periodic=FALSE,
          Dperiodic=periodic)
```

Arguments

expr	Function values to be differentiated. A function value table (object of class "fv", see fv.object).
which	Character vector identifying which columns of the table should be differentiated. Either a vector containing names of columns, or one of the wildcard strings "*" or "." explained below.
...	Extra arguments passed to smooth.spline to control the differentiation algorithm, if method="spline".
method	Differentiation method. A character string, partially matched to either "spline" or "numeric".
kinks	Optional vector of x values where the derivative is allowed to be discontinuous.
periodic	Logical value indicating whether the function expr is periodic.
Dperiodic	Logical value indicating whether the resulting derivative should be a periodic function.

Details

This command performs numerical differentiation on the function values in a function value table (object of class "fv"). The differentiation is performed either by smooth.spline or by a naive numerical difference algorithm.

The command deriv is generic. This is the method for objects of class "fv".

Differentiation is applied to every column (or to each of the selected columns) of function values in turn, using the function argument as the x coordinate and the selected column as the y coordinate. The original function values are then replaced by the corresponding derivatives.

The optional argument which specifies which of the columns of function values in expr will be differentiated. The default (indicated by the wildcard which="*") is to differentiate all function values, i.e.\ all columns except the function argument. Alternatively which="." designates the subset of function values that are displayed in the default plot. Alternatively which can be a character vector containing the names of columns of expr.

If the argument kinks is given, it should be a numeric vector giving the discontinuity points of the function: the value or values of the function argument at which the function is not differentiable. Differentiation will be performed separately on intervals between the discontinuity points.

If periodic=TRUE then the function expr is taken to be periodic, with period equal to the range of the function argument in expr. The resulting derivative is periodic.

If periodic=FALSE but Dperiodic=TRUE, then the *derivative* is assumed to be periodic. This would be appropriate if expr is the cumulative distribution function of an angular variable, for example.

Value

Another function value table (object of class "fv") of the same format.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

with.fv, fv.object, smooth.spline

Examples

```
G <- Gest(cells)
plot(deriv(G, which=".", spar=0.5))
A <- pairorient(redwood, 0.05, 0.15)
DA <- deriv(A, spar=0.6, Dperiodic=TRUE)
```

---

detpointprocfamilyfun *Construct a New Determinantal Point Process Model Family Function*

Description

Function to ease the implementation of a new determinantal point process model family.

Usage

```
detpointprocfamilyfun(kernel = NULL,
    specden = NULL, basis = "fourierbasis",
    convkernel = NULL, Kfun = NULL, valid = NULL, intensity = NULL,
    dim = 2, name = "User-defined", isotropic = TRUE, range = NULL,
    parbounds = NULL, specdenrange = NULL, startpar = NULL, ...)
```

Arguments

`kernel`	function specifying the kernel. May be set to `NULL`. See Details.
`specden`	function specifying the spectral density. May be set to `NULL`. See Details.
`basis`	character string giving the name of the basis. Defaults to the Fourier basis. See Details.
`convkernel`	function specifying the k-fold auto-convolution of the kernel. May be set to `NULL`. See Details.
`Kfun`	function specifying the K-function. May be set to `NULL`. See Details.
`valid`	function determining whether a given set of parameter values yields a valid model. May be set to `NULL`. See Examples.
`intensity`	character string specifying which parameter is the intensity in the model family. Should be `NULL` if the model family has no intensity parameter.
`dim`	character strig specifying which parameter is the dimension of the state space in this model family (if any). Alternatively a positive integer specifying the dimension.
`name`	character string giving the name of the model family used for printing.
`isotropic`	logical value indicating whether or not the model is isotropic.
`range`	function determining the interaction range of the model. May be set to `NULL`. See Examples.
`parbounds`	function determining the bounds for each model parameter when all other parameters are fixed. May be set to `NULL`. See Examples.
`specdenrange`	function specifying the the range of the spectral density if it is finite (only the case for very few models). May be set to `NULL`.
`startpar`	function determining starting values for parameters in any estimation algorithm. May be set to `NULL`. See Examples.
`...`	Additional arguments for inclusion in the returned model object. These are not checked in any way.

Details

A determinantal point process family is specified either in terms of a kernel (a positive semi-definite function, i.e. a covariance function) or a spectral density, or preferably both. One of these can be `NULL` if it is unknown, but not both. When both are supplied they must have the same arguments. The first argument gives the values at which the function should be evaluated. In general the function should accept an n by d matrix or `data.frame` specifying $n (>= 0)$ points in dimension d. If the model is isotropic it only needs to accept a non-negative valued numeric of length n. (In fact there is currently almost no support for non-isotropic models, so it is recommended not to specify such a model.) The name of this argument could be chosen freely, but x is recommended. The remaining arguments are the parameters of the model. If one of these is an intensity parameter the name should be mentioned in the argument `intensity`. If one of these specifies the dimension of the model it should be mentioned in the argument `dim`.

The kernel and spectral density is with respect to a specific set of basis functions, which is typically the Fourier basis. However this can be changed to any user-supplied basis in the argument `basis`. If such an alternative is supplied it must be the name of a function expecting the same arguments as `fourierbasis` and returning the results in the same form as `fourierbasis`.

If supplied, the arguments of convkernel must obey the following: first argument should be like the first argument of kernel and/or specden (see above). The second argument (preferably called `k`) should be the positive integer specifying how many times the auto-convolution is done (i.e. the k in k-fold auto-convolution). The remaining arguments must agree with the arguments of `kernel` and/or `specden` (see above).

If supplied, the arguments of `Kfun` should be like the arguments of `kernel` and `specden` (see above).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
  ## Example of how to define the Gauss family
exGauss <- detpointprocfamilyfun(
    name="Gaussian",
    kernel=function(x, lambda, alpha, d){
        lambda*exp(-(x/alpha)^2)
    },
    specden=function(x, lambda, alpha, d){
        lambda * (sqrt(pi)*alpha)^d * exp(-(x*alpha*pi)^2)
    },
    convkernel=function(x, k, lambda, alpha, d){
        logres <- k*log(lambda*pi*alpha^2) - log(pi*k*alpha^2) - x^2/(k*alpha^2)
        return(exp(logres))
    },
    Kfun = function(x, lambda, alpha, d){
        pi*x^2 - pi*alpha^2/2*(1-exp(-2*x^2/alpha^2))
    },
    valid=function(lambda, alpha, d){
        lambda>0 && alpha>0 && d>=1 && lambda <= (sqrt(pi)*alpha)^(-d)
    },
    isotropic=TRUE,
    intensity="lambda",
    dim="d",
    range=function(alpha, bound = .99){
        if(missing(alpha))
            stop("The parameter alpha is missing.")
        if(!(is.numeric(bound)&&bound>0&&bound<1))
            stop("Argument bound must be a numeric between 0 and 1.")
        return(alpha*sqrt(-log(sqrt(1-bound))))
    },
    parbounds=function(name, lambda, alpha, d){
        switch(name,
               lambda = c(0, (sqrt(pi)*alpha)^(-d)),
               alpha = c(0, lambda^(-1/d)/sqrt(pi)),
```

```
                stop("Parameter name misspecified")
                )
     },
     startpar=function(model, X){
         rslt <- NULL
         if("lambda" %in% model$freepar){
             lambda <- intensity(X)
             rslt <- c(rslt, "lambda" = lambda)
             model <- update(model, lambda=lambda)
         }
         if("alpha" %in% model$freepar){
             alpha <- .8*dppparbounds(model, "alpha")[2]
             rslt <- c(rslt, "alpha" = alpha)
         }
         return(rslt)
     }
     )
   exGauss
   m <- exGauss(lambda=100, alpha=.05, d=2)
   m
```

`dfbetas.ppm` *Parameter influence measure*

Description

Computes the deletion influence measure for each parameter in a fitted point process model.

Usage

```
## S3 method for class 'ppm'
dfbetas(model, ..., drop = FALSE, iScore=NULL,
iHessian=NULL, iArgs=NULL)
```

Arguments

`model`	Fitted point process model (object of class "ppm").
`...`	Ignored, except for the arguments `dimyx` and `eps` which are passed to `as.mask` to control the spatial resolution of the image of the density component.
`drop`	Logical. Whether to include (`drop=FALSE`) or exclude (`drop=TRUE`) contributions from quadrature points that were not used to fit the model.
`iScore,iHessian`	Components of the score vector and Hessian matrix for the irregular parameters, if required. See Details.
`iArgs`	List of extra arguments for the functions `iScore`, `iHessian` if required.

Details

Given a fitted spatial point process `model`, this function computes the influence measure for each parameter, as described in Baddeley, Chang and Song (2013).

This is a method for the generic function `dfbetas`.

The influence measure for each parameter θ is a signed measure in two-dimensional space. It consists of a discrete mass on each data point (i.e. each point in the point pattern to which the `model` was originally fitted) and a continuous density at all locations. The mass at a data point represents the change in the fitted value of the parameter θ that would occur if this data point were to be deleted. The density at other non-data locations represents the effect (on the fitted value of θ) of deleting these locations (and their associated covariate values) from the input to the fitting procedure.

If the point process model trend has irregular parameters that were fitted (using `ippm`) then the influence calculation requires the first and second derivatives of the log trend with respect to the irregular parameters. The argument `iScore` should be a list, with one entry for each irregular parameter, of R functions that compute the partial derivatives of the log trend (i.e. log intensity or log conditional intensity) with respect to each irregular parameter. The argument `iHessian` should be a list, with p^2 entries where p is the number of irregular parameters, of R functions that compute the second order partial derivatives of the log trend with respect to each pair of irregular parameters.

Value

An object of class `"msr"` representing a signed or vector-valued measure.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A. and Chang, Y.M. and Song, Y. (2013) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics* **40**, 86–104.

See Also

`leverage.ppm`, `influence.ppm`, `ppmInfluence`

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)

plot(dfbetas(fit))
plot(Smooth(dfbetas(fit)))
```

---

dg.envelope *Global Envelopes for Dao-Genton Test*

Description

Computes the global envelopes corresponding to the Dao-Genton test of goodness-of-fit.

Usage

```
dg.envelope(X, ...,
            nsim = 19, nsimsub=nsim-1, nrank = 1,
            alternative=c("two.sided", "less", "greater"),
            leaveout=1, interpolate = FALSE,
            savefuns=FALSE, savepatterns=FALSE,
            verbose = TRUE)
```

Arguments

`X`	Either a point pattern dataset (object of class `"ppp"`, `"lpp"` or `"pp3"`) or a fitted point process model (object of class `"ppm"`, `"kppm"` or `"slrm"`).
`...`	Arguments passed to `mad.test` or `envelope` to control the conduct of the test. Useful arguments include `fun` to determine the summary function, `rinterval` to determine the range of r values used in the test, and `verbose=FALSE` to turn off the messages.
`nsim`	Number of simulated patterns to be generated in the primary experiment.
`nsimsub`	Number of simulations in each basic test. There will be `nsim` repetitions of the basic test, each involving `nsimsub` simulated realisations, so there will be a total of `nsim * (nsimsub + 1)` simulations.
`nrank`	Integer. Rank of the envelope value amongst the `nsim` simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.
`alternative`	Character string determining whether the envelope corresponds to a two-sided test (`alternative="two.sided"`, the default) or a one-sided test with a lower critical boundary (`alternative="less"`) or a one-sided test with an upper critical boundary (`alternative="greater"`).
`leaveout`	Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.
`interpolate`	Logical value indicating whether to interpolate the distribution of the test statistic by kernel smoothing, as described in Dao and Genton (2014, Section 5).
`savefuns`	Logical flag indicating whether to save the simulated function values (from the first stage).
`savepatterns`	Logical flag indicating whether to save the simulated point patterns (from the first stage).
`verbose`	Logical value determining whether to print progress reports.

Details

Computes global simulation envelopes corresponding to the Dao-Genton (2014) adjusted Monte Carlo goodness-of-fit test. The envelopes are described in Baddeley et al (2015).

If `X` is a point pattern, the null hypothesis is CSR.

If `X` is a fitted model, the null hypothesis is that model.

Value

An object of class `"fv"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2015) Pushing the envelope: extensions of graphical Monte Carlo tests. Submitted for publication.

See Also

dg.test, mad.test, envelope

Examples

```
ns <- if(interactive()) 19 else 4
E <- dg.envelope(swedishpines, Lest, nsim=ns)
E
plot(E)
Eo <- dg.envelope(swedishpines, Lest, alternative="less", nsim=ns)
Ei <- dg.envelope(swedishpines, Lest, interpolate=TRUE, nsim=ns)
```

---

dg.progress *Progress Plot of Dao-Genton Test of Spatial Pattern*

Description

Generates a progress plot (envelope representation) of the Dao-Genton test for a spatial point pattern.

Usage

```
dg.progress(X, fun = Lest, ...,
            exponent = 2, nsim = 19, nsimsub = nsim - 1,
            nrank = 1, alpha, leaveout=1, interpolate = FALSE, rmin=0,
            savefuns = FALSE, savepatterns = FALSE, verbose=TRUE)
```

Arguments

X	Either a point pattern (object of class "ppp", "lpp" or other class), a fitted point process model (object of class "ppm", "kppm" or other class) or an envelope object (class "envelope").
fun	Function that computes the desired summary statistic for a point pattern.
...	Arguments passed to envelope. Useful arguments include alternative to specify one-sided or two-sided envelopes.

exponent	Positive number. The exponent of the L^p distance. See Details.
nsim	Number of repetitions of the basic test.
nsimsub	Number of simulations in each basic test. There will be nsim repetitions of the basic test, each involving nsimsub simulated realisations, so there will be a total of nsim * (nsimsub + 1) simulations.
nrank	Integer. The rank of the critical value of the Monte Carlo test, amongst the nsim simulated values. A rank of 1 means that the minimum and maximum simulated values will become the critical values for the test.
alpha	Optional. The significance level of the test. Equivalent to nrank/(nsim+1) where nsim is the number of simulations.
leaveout	Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.
interpolate	Logical value indicating how to compute the critical value. If interpolate=FALSE (the default), a standard Monte Carlo test is performed, and the critical value is the largest simulated value of the test statistic (if nrank=1) or the nrank-th largest (if nrank is another number). If interpolate=TRUE, kernel density estimation is applied to the simulated values, and the critical value is the upper alpha quantile of this estimated distribution.
rmin	Optional. Left endpoint for the interval of r values on which the test statistic is calculated.
savefuns	Logical value indicating whether to save the simulated function values (from the first stage).
savepatterns	Logical value indicating whether to save the simulated point patterns (from the first stage).
verbose	Logical value indicating whether to print progress reports.

Details

The Dao and Genton (2014) test for a spatial point pattern is described in dg.test. This test depends on the choice of an interval of distance values (the argument rinterval). A *progress plot* or *envelope representation* of the test (Baddeley et al, 2014) is a plot of the test statistic (and the corresponding critical value) against the length of the interval rinterval.

The command dg.progress effectively performs dg.test on X using all possible intervals of the form $[0, R]$, and returns the resulting values of the test statistic, and the corresponding critical values of the test, as a function of R.

The result is an object of class "fv" that can be plotted to obtain the progress plot. The display shows the test statistic (solid black line) and the test acceptance region (grey shading). If X is an envelope object, then some of the data stored in X may be re-used:

- If X is an envelope object containing simulated functions, and fun=NULL, then the code will re-use the simulated functions stored in X.
- If X is an envelope object containing simulated point patterns, then fun will be applied to the stored point patterns to obtain the simulated functions. If fun is not specified, it defaults to Lest.
- Otherwise, new simulations will be performed, and fun defaults to Lest.

If the argument rmin is given, it specifies the left endpoint of the interval defining the test statistic: the tests are performed using intervals $[r_{\min}, R]$ where $R \geq r_{\min}$.

The argument leaveout specifies how to calculate the discrepancy between the summary function for the data and the nominal reference value, when the reference value must be estimated by simulation. The values leaveout=0 and leaveout=1 are both algebraically equivalent (Baddeley et al, 2014, Appendix) to computing the difference observed - reference where the reference is the mean of simulated values. The value leaveout=2 gives the leave-two-out discrepancy proposed by Dao and Genton (2014).

Value

An object of class "fv" that can be plotted to obtain the progress plot.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A., Diggle, P., Hardegen, A., Lawrence, T., Milne, R. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84** (3) 477–489.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2015) Pushing the envelope: extensions of graphical Monte Carlo tests. Submitted for publication.

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

See Also

dg.test, dclf.progress

Examples

```
ns <- if(interactive()) 19 else 5
plot(dg.progress(cells, nsim=ns))
```

---

dg.sigtrace *Significance Trace of Dao-Genton Test*

Description

Generates a Significance Trace of the Dao and Genton (2014) test for a spatial point pattern.

Usage

```
dg.sigtrace(X, fun = Lest, ...,
            exponent = 2, nsim = 19, nsimsub = nsim - 1,
            alternative = c("two.sided", "less", "greater"),
            rmin=0, leaveout=1,
            interpolate = FALSE, confint = TRUE, alpha = 0.05,
            savefuns=FALSE, savepatterns=FALSE, verbose=FALSE)
```

Arguments

`X`	Either a point pattern (object of class "ppp", "lpp" or other class), a fitted point process model (object of class "ppm", "kppm" or other class) or an envelope object (class "envelope").
`fun`	Function that computes the desired summary statistic for a point pattern.
`...`	Arguments passed to `envelope`.
`exponent`	Positive number. Exponent used in the test statistic. Use `exponent=2` for the Diggle-Cressie-Loosmore-Ford test, and `exponent=Inf` for the Maximum Absolute Deviation test. See Details.
`nsim`	Number of repetitions of the basic test.
`nsimsub`	Number of simulations in each basic test. There will be `nsim` repetitions of the basic test, each involving `nsimsub` simulated realisations, so there will be a total of `nsim * (nsimsub + 1)` simulations.
`alternative`	Character string specifying the alternative hypothesis. The default (`alternative="two.sided"`) is that the true value of the summary function is not equal to the theoretical value postulated under the null hypothesis. If `alternative="less"` the alternative hypothesis is that the true value of the summary function is lower than the theoretical value.
`rmin`	Optional. Left endpoint for the interval of r values on which the test statistic is calculated.
`leaveout`	Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.
`interpolate`	Logical value indicating whether to interpolate the distribution of the test statistic by kernel smoothing, as described in Dao and Genton (2014, Section 5).
`confint`	Logical value indicating whether to compute a confidence interval for the 'true' p-value.
`alpha`	Significance level to be plotted (this has no effect on the calculation but is simply plotted as a reference value).
`savefuns`	Logical flag indicating whether to save the simulated function values (from the first stage).
`savepatterns`	Logical flag indicating whether to save the simulated point patterns (from the first stage).
`verbose`	Logical flag indicating whether to print progress reports.

Details

The Dao and Genton (2014) test for a spatial point pattern is described in `dg.test`. This test depends on the choice of an interval of distance values (the argument `rinterval`). A *significance trace* (Bowman and Azzalini, 1997; Baddeley et al, 2014, 2015) of the test is a plot of the p-value obtained from the test against the length of the interval `rinterval`.

The command `dg.sigtrace` effectively performs `dg.test` on `X` using all possible intervals of the form $[0, R]$, and returns the resulting p-values as a function of R.

The result is an object of class "fv" that can be plotted to obtain the significance trace. The plot shows the Dao-Genton adjusted p-value (solid black line), the critical value `0.05` (dashed red line), and a pointwise 95% confidence band (grey shading) for the 'true' (Neyman-Pearson) p-value. The confidence band is based on the Agresti-Coull (1998) confidence interval for a binomial proportion.

If X is an envelope object and fun=NULL then the code will re-use the simulated functions stored in X.

If the argument rmin is given, it specifies the left endpoint of the interval defining the test statistic: the tests are performed using intervals $[r_{\min}, R]$ where $R \geq r_{\min}$.

The argument leaveout specifies how to calculate the discrepancy between the summary function for the data and the nominal reference value, when the reference value must be estimated by simulation. The values leaveout=0 and leaveout=1 are both algebraically equivalent (Baddeley et al, 2014, Appendix) to computing the difference observed - reference where the reference is the mean of simulated values. The value leaveout=2 gives the leave-two-out discrepancy proposed by Dao and Genton (2014).

Value

An object of class "fv" that can be plotted to obtain the significance trace.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Agresti, A. and Coull, B.A. (1998) Approximate is better than "Exact" for interval estimation of binomial proportions. *American Statistician* **52**, 119–126.

Baddeley, A., Diggle, P., Hardegen, A., Lawrence, T., Milne, R. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84**(3) 477–489.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2015) Pushing the envelope: extensions of graphical Monte Carlo tests. Submitted for publication.

Bowman, A.W. and Azzalini, A. (1997) *Applied smoothing techniques for data analysis: the kernel approach with S-Plus illustrations*. Oxford University Press, Oxford.

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

See Also

dg.test for the Dao-Genton test, dclf.sigtrace for significance traces of other tests.

Examples

```
ns <- if(interactive()) 19 else 5
plot(dg.sigtrace(cells, nsim=ns))
```

dg.test *Dao-Genton Adjusted Goodness-Of-Fit Test*

Description

Performs the Dao and Genton (2014) adjusted goodness-of-fit test of spatial pattern.

Usage

```
dg.test(X, ...,
        exponent = 2, nsim=19, nsimsub=nsim-1,
        alternative=c("two.sided", "less", "greater"),
        reuse = TRUE, leaveout=1, interpolate = FALSE,
        savefuns=FALSE, savepatterns=FALSE,
        verbose = TRUE)
```

Arguments

`X`	Either a point pattern dataset (object of class "ppp", "lpp" or "pp3") or a fitted point process model (object of class "ppm", "kppm", "lppm" or "slrm").
`...`	Arguments passed to `dclf.test` or `mad.test` or `envelope` to control the conduct of the test. Useful arguments include `fun` to determine the summary function, `rinterval` to determine the range of r values used in the test, and `use.theory` described under Details.
`exponent`	Exponent used in the test statistic. Use `exponent=2` for the Diggle-Cressie-Loosmore-Ford test, and `exponent=Inf` for the Maximum Absolute Deviation test.
`nsim`	Number of repetitions of the basic test.
`nsimsub`	Number of simulations in each basic test. There will be `nsim` repetitions of the basic test, each involving `nsimsub` simulated realisations, so there will be a total of `nsim * (nsimsub + 1)` simulations.
`alternative`	Character string specifying the alternative hypothesis. The default (`alternative="two.sided"`) is that the true value of the summary function is not equal to the theoretical value postulated under the null hypothesis. If `alternative="less"` the alternative hypothesis is that the true value of the summary function is lower than the theoretical value.
`reuse`	Logical value indicating whether to re-use the first stage simulations at the second stage, as described by Dao and Genton (2014).
`leaveout`	Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.
`interpolate`	Logical value indicating whether to interpolate the distribution of the test statistic by kernel smoothing, as described in Dao and Genton (2014, Section 5).
`savefuns`	Logical flag indicating whether to save the simulated function values (from the first stage).
`savepatterns`	Logical flag indicating whether to save the simulated point patterns (from the first stage).
`verbose`	Logical value indicating whether to print progress reports.

Details

Performs the Dao-Genton (2014) adjusted Monte Carlo goodness-of-fit test, in the equivalent form described by Baddeley et al (2014).

If `X` is a point pattern, the null hypothesis is CSR.

If `X` is a fitted model, the null hypothesis is that model.

The argument `use.theory` passed to `envelope` determines whether to compare the summary function for the data to its theoretical value for CSR (`use.theory=TRUE`) or to the sample mean of simulations from CSR (`use.theory=FALSE`).

The argument `leaveout` specifies how to calculate the discrepancy between the summary function for the data and the nominal reference value, when the reference value must be estimated by simulation. The values `leaveout=0` and `leaveout=1` are both algebraically equivalent (Baddeley et al, 2014, Appendix) to computing the difference `observed - reference` where the `reference` is the mean of simulated values. The value `leaveout=2` gives the leave-two-out discrepancy proposed by Dao and Genton (2014).

The Dao-Genton test is biased when the significance level is very small (small p-values are not reliable) and we recommend `bits.test` in this case.

Value

A hypothesis test (object of class `"htest"` which can be printed to show the outcome of the test.

Author(s)

Adrian Baddeley, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<r.turner@auckland.ac.n` and Ege Rubak `<rubak@math.aau.dk>`.

References

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

Baddeley, A., Diggle, P.J., Hardegen, A., Lawrence, T., Milne, R.K. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84** (3) 477–489.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2017) On two-stage Monte Carlo tests of composite hypotheses. *Computational Statistics and Data Analysis*, in press.

See Also

`bits.test`, `dclf.test`, `mad.test`

Examples

```
ns <- if(interactive()) 19 else 4
dg.test(cells, nsim=ns)
dg.test(cells, alternative="less", nsim=ns)
dg.test(cells, nsim=ns, interpolate=TRUE)
```

diagnose.ppm *Diagnostic Plots for Fitted Point Process Model*

Description

Given a point process model fitted to a point pattern, produce diagnostic plots based on residuals.

Usage

```
diagnose.ppm(object, ..., type="raw", which="all", sigma=NULL,
             rbord=reach(object), cumulative=TRUE,
             plot.it=TRUE, rv = NULL,
             compute.sd=is.poisson(object), compute.cts=TRUE,
             envelope=FALSE, nsim=39, nrank=1,
             typename, check=TRUE, repair=TRUE,
             oldstyle=FALSE, splineargs=list(spar=0.5))

## S3 method for class 'diagppm'
plot(x, ..., which,
             plot.neg=c("image", "discrete", "contour", "imagecontour"),
             plot.smooth=c("imagecontour", "image", "contour", "persp"),
             plot.sd, spacing=0.1, outer=3,
             srange=NULL, monochrome=FALSE, main=NULL)
```

Arguments

`object`	The fitted point process model (an object of class "ppm") for which diagnostics should be produced. This object is usually obtained from ppm.
`type`	String indicating the type of residuals or weights to be used. Current options are "eem" for the Stoyan-Grabarnik exponential energy weights, "raw" for the raw residuals, "inverse" for the inverse-lambda residuals, and "pearson" for the Pearson residuals. A partial match is adequate.
`which`	Character string or vector indicating the choice(s) of plots to be generated. Options are "all", "marks", "smooth", "x", "y" and "sum". Multiple choices may be given but must be matched exactly. See Details.
`sigma`	Bandwidth for kernel smoother in "smooth" option.
`rbord`	Width of border to avoid edge effects. The diagnostic calculations will be confined to those points of the data pattern which are at least `rbord` units away from the edge of the window. (An infinite value of `rbord` will be ignored.)
`cumulative`	Logical flag indicating whether the lurking variable plots for the x and y coordinates will be the plots of cumulative sums of marks (`cumulative=TRUE`) or the plots of marginal integrals of the smoothed residual field (`cumulative=FALSE`).
`plot.it`	Logical value indicating whether plots should be shown. If `plot.it=FALSE`, the computed diagnostic quantities are returned without plotting them.
`plot.neg`	String indicating how the density part of the residual measure should be plotted.
`plot.smooth`	String indicating how the smoothed residual field should be plotted.

compute.sd,plot.sd
: Logical values indicating whether error bounds should be computed and added to the "x" and "y" plots. The default is TRUE for Poisson models and FALSE for non-Poisson models. See Details.

envelope,nsim,nrank
: Arguments passed to lurking in order to plot simulation envelopes for the lurking variable plots.

rv
: Usually absent. Advanced use only. If this argument is present, the values of the residuals will not be calculated from the fitted model object but will instead be taken directly from rv.

spacing
: The spacing between plot panels (when a four-panel plot is generated) expressed as a fraction of the width of the window of the point pattern.

outer
: The distance from the outermost line of text to the nearest plot panel, expressed as a multiple of the spacing between plot panels.

srange
: Vector of length 2 that will be taken as giving the range of values of the smoothed residual field, when generating an image plot of this field. This is useful if you want to generate diagnostic plots for two different fitted models using the same colour map.

monochrome
: Flag indicating whether images should be displayed in greyscale (suitable for publication) or in colour (suitable for the screen). The default is to display in colour.

check
: Logical value indicating whether to check the internal format of object. If there is any possibility that this object has been restored from a dump file, or has otherwise lost track of the environment where it was originally computed, set check=TRUE.

repair
: Logical value indicating whether to repair the internal format of object, if it is found to be damaged.

oldstyle
: Logical flag indicating whether error bounds should be plotted using the approximation given in the original paper (oldstyle=TRUE), or using the correct asymptotic formula (oldstyle=FALSE).

splineargs
: Argument passed to lurking to control the smoothing in the lurking variable plot.

x
: The value returned from a previous call to diagnose.ppm. An object of class "diagppm".

typename
: String to be used as the name of the residuals.

main
: Main title for the plot.

...
: Extra arguments, controlling either the resolution of the smoothed image (passed from diagnose.ppm to density.ppp) or the appearance of the plots (passed from diagnose.ppm to plot.diagppm and from plot.diagppm to plot.default).

compute.cts
: Advanced use only.

Details

The function diagnose.ppm generates several diagnostic plots for a fitted point process model. The plots display the residuals from the fitted model (Baddeley et al, 2005) or alternatively the 'exponential energy marks' (Stoyan and Grabarnik, 1991). These plots can be used to assess goodness-of-fit, to identify outliers in the data, and to reveal departures from the fitted model. See also the companion function qqplot.ppm.

The argument `object` must be a fitted point process model (object of class `"ppm"`) typically produced by the maximum pseudolikelihood fitting algorithm `ppm`).

The argument `type` selects the type of residual or weight that will be computed. Current options are:

`"eem"`: exponential energy marks (Stoyan and Grabarnik, 1991) computed by `eem`. These are positive weights attached to the data points (i.e. the points of the point pattern dataset to which the model was fitted). If the fitted model is correct, then the sum of these weights for all data points in a spatial region B has expected value equal to the area of B. See `eem` for further explanation.

`"raw"`, `"inverse"` or `"pearson"`: point process residuals (Baddeley et al, 2005) computed by the function `residuals.ppm`. These are residuals attached both to the data points and to some other points in the window of observation (namely, to the dummy points of the quadrature scheme used to fit the model). If the fitted model is correct, then the sum of the residuals in a spatial region B has mean zero. The options are

- `"raw"`: the raw residuals;
- `"inverse"`: the 'inverse-lambda' residuals, a counterpart of the exponential energy weights;
- `"pearson"`: the Pearson residuals.

See `residuals.ppm` for further explanation.

The argument `which` selects the type of plot that is produced. Options are:

`"marks"`: plot the residual measure. For the exponential energy weights (`type="eem"`) this displays circles centred at the points of the data pattern, with radii proportional to the exponential energy weights. For the residuals (`type="raw"`, `type="inverse"` or `type="pearson"`) this again displays circles centred at the points of the data pattern with radii proportional to the (positive) residuals, while the plotting of the negative residuals depends on the argument `plot.neg`. If `plot.neg="image"` then the negative part of the residual measure, which is a density, is plotted as a colour image. If `plot.neg="discrete"` then the discretised negative residuals (obtained by approximately integrating the negative density using the quadrature scheme of the fitted model) are plotted as squares centred at the dummy points with side lengths proportional to the (negative) residuals. [To control the size of the circles and squares, use the argument `maxsize`.]

`"smooth"`: plot a kernel-smoothed version of the residual measure. Each data or dummy point is taken to have a 'mass' equal to its residual or exponential energy weight. (Note that residuals can be negative). This point mass is then replaced by a bivariate isotropic Gaussian density with standard deviation `sigma`. The value of the smoothed residual field at any point in the window is the sum of these weighted densities. If the fitted model is correct, this smoothed field should be flat, and its height should be close to 0 (for the residuals) or 1 (for the exponential energy weights). The field is plotted either as an image, contour plot or perspective view of a surface, according to the argument `plot.smooth`. The range of values of the smoothed field is printed if the option `which="sum"` is also selected.

`"x"`: produce a 'lurking variable' plot for the x coordinate. This is a plot of $h(x)$ against x (solid lines) and of $E(h(x))$ against x (dashed lines), where $h(x)$ is defined below, and $E(h(x))$ denotes the expectation of $h(x)$ assuming the fitted model is true.

- if `cumulative=TRUE` then $h(x)$ is the cumulative sum of the weights or residuals for all points which have X coordinate less than or equal to x. For the residuals $E(h(x)) = 0$, and for the exponential energy weights $E(h(x))$ = area of the subset of the window to the left of the line $X = x$.

- if `cumulative=FALSE` then $h(x)$ is the marginal integral of the smoothed residual field (see the case `which="smooth"` described above) on the x axis. This is approximately the derivative of the plot for `cumulative=TRUE`. The value of $h(x)$ is computed by summing the values of the smoothed residual field over all pixels with the given x coordinate. For the residuals $E(h(x)) = 0$, and for the exponential energy weights $E(h(x)) =$ length of the intersection between the observation window and the line $X = x$.

If `plot.sd = TRUE`, then superimposed on the lurking variable plot are the pointwise two-standard-deviation error limits for $h(x)$ calculated for the inhomogeneous Poisson process. The default is `plot.sd = TRUE` for Poisson models and `plot.sd = FALSE` for non-Poisson models.

`"y"`: produce a similar lurking variable plot for the y coordinate.

`"sum"`: print the sum of the weights or residuals for all points in the window (clipped by a margin `rbord` if required) and the area of the same window. If the fitted model is correct the sum of the exponential energy weights should equal the area of the window, while the sum of the residuals should equal zero. Also print the range of values of the smoothed field displayed in the `"smooth"` case.

`"all"`: All four of the diagnostic plots listed above are plotted together in a two-by-two display. Top left panel is `"marks"` plot. Bottom right panel is `"smooth"` plot. Bottom left panel is `"x"` plot. Top right panel is `"y"` plot, rotated 90 degrees.

The argument `rbord` ensures there are no edge effects in the computation of the residuals. The diagnostic calculations will be confined to those points of the data pattern which are at least `rbord` units away from the edge of the window. The value of `rbord` should be greater than or equal to the range of interaction permitted in the model.

By default, the two-standard-deviation limits are calculated from the exact formula for the asymptotic variance of the residuals under the asymptotic normal approximation, equation (37) of Baddeley et al (2006). However, for compatibility with the original paper of Baddeley et al (2005), if `oldstyle=TRUE`, the two-standard-deviation limits are calculated using the innovation variance, an over-estimate of the true variance of the residuals. (However, see the section about Replicated Data).

The argument `rv` would normally be used only by experts. It enables the user to substitute arbitrary values for the residuals or marks, overriding the usual calculations. If `rv` is present, then instead of calculating the residuals from the fitted model, the algorithm takes the residuals from the object `rv`, and plots them in the manner appropriate to the type of residual or mark selected by `type`. If `type ="eem"` then `rv` should be similar to the return value of `eem`, namely, a numeric vector of length equal to the number of points in the original data point pattern. Otherwise, `rv` should be similar to the return value of `residuals.ppm`, that is, it should be an object of class `"msr"` (see `msr`) representing a signed measure.

The return value of `diagnose.ppm` is an object of class `"diagppm"`. The `plot` method for this class is documented here. There is also a `print` method. See the Examples.

In `plot.diagppm`, if a four-panel diagnostic plot is produced (the default), then the extra arguments `xlab`, `ylab`, `rlab` determine the text labels for the x and y coordinates and the residuals, respectively. The undocumented arguments `col.neg` and `col.smooth` control the colour maps used in the top left and bottom right panels respectively.

See also the companion functions `qqplot.ppm`, which produces a Q-Q plot of the residuals, and `lurking`, which produces lurking variable plots for any spatial covariate.

Value

An object of class `"diagppm"` which contains the coordinates needed to reproduce the selected plots. This object can be plotted using `plot.diagppm` and printed using `print.diagppm`.

Replicated Data

Note that if `object` is a model that was obtained by first fitting a model to replicated point pattern data using `mppm` and then using `subfits` to extract a model for one of the individual point patterns, then the variance calculations are only implemented for the innovation variance (`oldstyle=TRUE`) and this is the default in such cases.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

Baddeley, A., Møller, J. and Pakes, A.G. (2008) Properties of residuals for spatial point processes. *Annals of the Institute of Statistical Mathematics* **60**, 627–649.

Stoyan, D. and Grabarnik, P. (1991) Second-order characteristics for stochastic structures connected with Gibbs point processes. *Mathematische Nachrichten*, 151:95–100.

See Also

`residuals.ppm`, `eem`, `ppm.object`, `qqplot.ppm`, `lurking`, `ppm`

Examples

```
    fit <- ppm(cells ~x, Strauss(r=0.15))
    diagnose.ppm(fit)
    ## Not run:
    diagnose.ppm(fit, type="pearson")

## End(Not run)

    diagnose.ppm(fit, which="marks")

    diagnose.ppm(fit, type="raw", plot.neg="discrete")

    diagnose.ppm(fit, type="pearson", which="smooth")

    # save the diagnostics and plot them later
    u <- diagnose.ppm(fit, rbord=0.15, plot.it=FALSE)
    ## Not run:
    plot(u)
    plot(u, which="marks")

## End(Not run)
```

`diameter` *Diameter of an Object*

Description

Computes the diameter of an object such as a two-dimensional window or three-dimensional box.

Usage

```
diameter(x)
```

Arguments

x	A window or other object whose diameter will be computed.

Details

This function computes the diameter of an object such as a two-dimensional window or a three-dimensional box. The diameter is the maximum distance between any two points in the object.

The function `diameter` is generic, with methods for the class `"owin"` (two-dimensional windows), `"box3"` (three-dimensional boxes), `"boxx"` (multi-dimensional boxes) and `"linnet"` (linear networks).

Value

The numerical value of the diameter of the object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`diameter.owin`, `diameter.box3`, `diameter.boxx`, `diameter.linnet`

---

diameter.box3 *Geometrical Calculations for Three-Dimensional Box*

Description

Calculates the volume, diameter, shortest side, side lengths, or eroded volume of a three-dimensional box.

Usage

```
## S3 method for class 'box3'
diameter(x)

## S3 method for class 'box3'
volume(x)

shortside(x)
sidelengths(x)
eroded.volumes(x, r)

## S3 method for class 'box3'
shortside(x)
```

```
## S3 method for class 'box3'
sidelengths(x)

## S3 method for class 'box3'
eroded.volumes(x, r)
```

Arguments

x	Three-dimensional box (object of class "box3").
r	Numeric value or vector of numeric values for which eroded volumes should be calculated.

Details

diameter.box3 computes the diameter of the box. volume.box3 computes the volume of the box. shortside.box3 finds the shortest of the three side lengths of the box. sidelengths.box3 returns all three side lengths of the box.

eroded.volumes computes, for each entry r[i], the volume of the smaller box obtained by removing a slab of thickness r[i] from each face of the box. This smaller box is the subset consisting of points that lie at least r[i] units away from the boundary of the box.

Value

For diameter.box3, shortside.box3 and volume.box3, a single numeric value. For sidelengths.box3, a vector of three numbers. For eroded.volumes, a numeric vector of the same length as r.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

as.box3

Examples

```
X <- box3(c(0,10),c(0,10),c(0,5))
diameter(X)
volume(X)
sidelengths(X)
shortside(X)
hd <- shortside(X)/2
eroded.volumes(X, seq(0,hd, length=10))
```

diameter.boxx *Geometrical Calculations for Multi-Dimensional Box*

Description

Calculates the volume, diameter, shortest side, side lengths, or eroded volume of a multi-dimensional box.

Usage

```
## S3 method for class 'boxx'
diameter(x)

## S3 method for class 'boxx'
volume(x)

## S3 method for class 'boxx'
shortside(x)

## S3 method for class 'boxx'
sidelengths(x)

## S3 method for class 'boxx'
eroded.volumes(x, r)
```

Arguments

x	Multi-dimensional box (object of class "boxx").
r	Numeric value or vector of numeric values for which eroded volumes should be calculated.

Details

`diameter.boxx`, `volume.boxx` and `shortside.boxx` compute the diameter, volume and shortest side length of the box. `sidelengths.boxx` returns the lengths of each side of the box.

`eroded.volumes.boxx` computes, for each entry `r[i]`, the volume of the smaller box obtained by removing a slab of thickness `r[i]` from each face of the box. This smaller box is the subset consisting of points that lie at least `r[i]` units away from the boundary of the box.

Value

For `diameter.boxx`, `shortside.boxx` and `volume.boxx`, a single numeric value. For `sidelengths.boxx`, a numeric vector of length equal to the number of spatial dimensions. For `eroded.volumes.boxx`, a numeric vector of the same length as `r`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

boxx

Examples

```
X <- boxx(c(0,10),c(0,10),c(0,5),c(0,2))
diameter(X)
volume(X)
shortside(X)
sidelengths(X)
hd <- shortside(X)/2
eroded.volumes(X, seq(0,hd, length=10))
```

---

diameter.linnet *Diameter and Bounding Radius of a Linear Network*

Description

Compute the diameter or bounding radius of a linear network measured using the shortest path distance.

Usage

```
## S3 method for class 'linnet'
diameter(x)

## S3 method for class 'linnet'
boundingradius(x, ...)
```

Arguments

x	Linear network (object of class "linnet").
...	Ignored.

Details

The diameter of a linear network (in the shortest path distance) is the maximum value of the shortest-path distance between any two points u and v on the network.

The bounding radius of a linear network (in the shortest path distance) is the minimum value, over all points u on the network, of the maximum shortest-path distance from u to another point v on the network.

The functions boundingradius and diameter are generic; the functions boundingradius.linnet and diameter.linnet are the methods for objects of class linnet.

Value

A single numeric value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

boundingradius, diameter, linnet

Examples

```
diameter(simplenet)
boundingradius(simplenet)
```

---

diameter.owin *Diameter of a Window*

Description

Computes the diameter of a window.

Usage

```
## S3 method for class 'owin'
diameter(x)
```

Arguments

x A window whose diameter will be computed.

Details

This function computes the diameter of a window of arbitrary shape, i.e. the maximum distance between any two points in the window.

The argument x should be a window (an object of class "owin", see owin.object for details) or can be given in any format acceptable to as.owin().

The function diameter is generic. This function is the method for the class "owin".

Value

The numerical value of the diameter of the window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

area.owin, perimeter, edges, owin, as.owin

Examples

```
w <- owin(c(0,1),c(0,1))
diameter(w)
# returns sqrt(2)
data(letterR)
diameter(letterR)
```

DiggleGatesStibbard *Diggle-Gates-Stibbard Point Process Model*

Description

Creates an instance of the Diggle-Gates-Stibbard point process model which can then be fitted to point pattern data.

Usage

```
DiggleGatesStibbard(rho)
```

Arguments

rho	Interaction range

Details

Diggle, Gates and Stibbard (1987) proposed a pairwise interaction point process in which each pair of points separated by a distance d contributes a factor $e(d)$ to the probability density, where

$$e(d) = \sin^2\left(\frac{\pi d}{2\rho}\right)$$

for $d < \rho$, and $e(d)$ is equal to 1 for $d \geq \rho$.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Diggle-Gates-Stibbard pairwise interaction is yielded by the function `DiggleGatesStibbard()`. See the examples below.

Note that this model does not have any regular parameters (as explained in the section on Interaction Parameters in the help file for `ppm`). The parameter ρ is not estimated by `ppm`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the Diggle-Gates-Stibbard process with interaction range `rho`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

Diggle, P.J., Gates, D.J., and Stibbard, A. (1987) A nonparametric estimator for pairwise-interaction point processes. Biometrika **74**, 763 – 770. *Scandinavian Journal of Statistics* **21**, 359–373.

See Also

ppm, pairwise.family, DiggleGratton, rDGS, ppm.object

Examples

```
DiggleGatesStibbard(0.02)
# prints a sensible description of itself

## Not run:
ppm(cells ~1, DiggleGatesStibbard(0.05))
# fit the stationary D-G-S process to `cells'

## End(Not run)

ppm(cells ~ polynom(x,y,3), DiggleGatesStibbard(0.05))
# fit a nonstationary D-G-S process
# with log-cubic polynomial trend
```

DiggleGratton *Diggle-Gratton model*

Description

Creates an instance of the Diggle-Gratton pairwise interaction point process model, which can then be fitted to point pattern data.

Usage

```
DiggleGratton(delta=NA, rho)
```

Arguments

delta	lower threshold δ
rho	upper threshold ρ

Details

Diggle and Gratton (1984, pages 208-210) introduced the pairwise interaction point process with pair potential $h(t)$ of the form

$$h(t) = \left(\frac{t-\delta}{\rho-\delta}\right)^{\kappa} \qquad \text{if } \delta \le t \le \rho$$

with $h(t) = 0$ for $t < \delta$ and $h(t) = 1$ for $t > \rho$. Here δ, ρ and κ are parameters.

Note that we use the symbol κ where Diggle and Gratton (1984) and Diggle, Gates and Stibbard (1987) use β, since in **spatstat** we reserve the symbol β for an intensity parameter.

The parameters must all be nonnegative, and must satisfy $\delta \le \rho$.

The potential is inhibitory, i.e.\ this model is only appropriate for regular point patterns. The strength of inhibition increases with κ. For $\kappa = 0$ the model is a hard core process with hard core radius δ. For $\kappa = \infty$ the model is a hard core process with hard core radius ρ.

The irregular parameters δ, ρ must be given in the call to `DiggleGratton`, while the regular parameter κ will be estimated.

If the lower threshold `delta` is missing or `NA`, it will be estimated from the data when `ppm` is called. The estimated value of `delta` is the minimum nearest neighbour distance multiplied by $n/(n+1)$, where n is the number of data points.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

References

Diggle, P.J., Gates, D.J. and Stibbard, A. (1987) A nonparametric estimator for pairwise-interaction point processes. *Biometrika* **74**, 763 – 770.

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

See Also

`ppm`, `ppm.object`, `Pairwise`

Examples

```
ppm(cells ~1, DiggleGratton(0.05, 0.1))
```

`dilated.areas` *Areas of Morphological Dilations*

Description

Computes the areas of successive morphological dilations.

Usage

```
dilated.areas(X, r, W=as.owin(X), ..., constrained=TRUE, exact = FALSE)
```

Arguments

`X`	Object to be dilated. A point pattern (object of class `"ppp"`), a line segment pattern (object of class `"psp"`), or a window (object of class `"owin"`).
`r`	Numeric vector of radii for the dilations.
`W`	Window (object of class `"owin"`) inside which the areas will be computed, if `constrained=TRUE`.
`...`	Arguments passed to `distmap` to control the pixel resolution, if `exact=FALSE`.
`constrained`	Logical flag indicating whether areas should be restricted to the window `W`.
`exact`	Logical flag indicating whether areas should be computed using analytic geometry (which is slower but more accurate). Currently available only when `X` is a point pattern.

Details

This function computes the areas of the dilations of X by each of the radii r[i]. Areas may also be computed inside a specified window W.

The morphological dilation of a set X by a distance $r > 0$ is the subset consisting of all points x such that the distance from x to X is less than or equal to r.

When X is a point pattern, the dilation by a distance r is the union of discs of radius r centred at the points of X.

The argument r should be a vector of nonnegative numbers.

If exact=TRUE and if X is a point pattern, then the areas are computed using analytic geometry, which is slower but much more accurate. Otherwise the computation is performed using distmap.

To compute the dilated object itself, use dilation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin, as.owin, dilation, eroded.areas

Examples

```
X <- runifpoint(10)
a <- dilated.areas(X, c(0.1,0.2), W=square(1), exact=TRUE)
```

---

dilation *Morphological Dilation*

Description

Perform morphological dilation of a window, a line segment pattern or a point pattern

Usage

```
 dilation(w, r, ...)
 ## S3 method for class 'owin'
dilation(w, r, ..., polygonal=NULL, tight=TRUE)
 ## S3 method for class 'ppp'
dilation(w, r, ..., polygonal=TRUE, tight=TRUE)
 ## S3 method for class 'psp'
dilation(w, r, ..., polygonal=TRUE, tight=TRUE)
```

Arguments

- `w` — A window (object of class "owin" or a line segment pattern (object of class "psp") or a point pattern (object of class "ppp").
- `r` — positive number: the radius of dilation.
- `...` — extra arguments passed to `as.mask` controlling the pixel resolution, if the pixel approximation is used; or passed to `disc` if the polygonal approximation is used.
- `polygonal` — Logical flag indicating whether to compute a polygonal approximation to the dilation (`polygonal=TRUE`) or a pixel grid approximation (`polygonal=FALSE`).
- `tight` — Logical flag indicating whether the bounding frame of the window should be taken as the smallest rectangle enclosing the dilated region (`tight=TRUE`), or should be the dilation of the bounding frame of `w` (`tight=FALSE`).

Details

The morphological dilation of a set W by a distance $r > 0$ is the set consisting of all points lying at most r units away from W. Effectively, dilation adds a margin of width r onto the set W.

If `polygonal=TRUE` then a polygonal approximation to the dilation is computed. If `polygonal=FALSE` then a pixel approximation to the dilation is computed from the distance map of `w`. The arguments "..." are passed to `as.mask` to control the pixel resolution.

When `w` is a window, the default (when `polygonal=NULL`) is to compute a polygonal approximation if `w` is a rectangle or polygonal window, and to compute a pixel approximation if `w` is a window of type "mask".

Value

If `r > 0`, an object of class "owin" representing the dilated region. If `r=0`, the result is identical to `w`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`erosion` for the opposite operation.

`dilationAny` for morphological dilation using any shape.

`owin`, `as.owin`

Examples

```
plot(dilation(letterR, 0.2))
plot(letterR, add=TRUE, lwd=2, border="red")

X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(dilation(X, 0.1))
plot(X, add=TRUE, col="red")
```

`dim.detpointprocfamily` *Dimension of Determinantal Point Process Model*

Description

Extracts the dimension of a determinantal point process model.

Usage

```
## S3 method for class 'detpointprocfamily'
dim(x)
```

Arguments

x	object of class "detpointprocfamily".

Value

A numeric (or NULL if the dimension of the model is unspecified).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

`dimhat` *Estimate Dimension of Central Subspace*

Description

Given the kernel matrix that characterises a central subspace, this function estimates the dimension of the subspace.

Usage

```
dimhat(M)
```

Arguments

M	Kernel of subspace. A symmetric, non-negative definite, numeric matrix, typically obtained from sdr.

Details

This function computes the maximum descent estimate of the dimension of the central subspace with a given kernel matrix `M`.

The matrix `M` should be the kernel matrix of a central subspace, which can be obtained from `sdr`. It must be a symmetric, non-negative-definite, numeric matrix.

The algorithm finds the eigenvalues $\lambda_1 \geq \ldots \geq \lambda_n$ of M, and then determines the index k for which λ_k/λ_{k-1} is greatest.

Value

A single integer giving the estimated dimension.

Author(s)

Matlab original by Yongtao Guan, translated to R by Suman Rakshit.

References

Guan, Y. and Wang, H. (2010) Sufficient dimension reduction for spatial point processes directed by Gaussian random fields. *Journal of the Royal Statistical Society, Series B*, **72**, 367–387.

See Also

`sdr`, `subspaceDistance`

`dirichlet` *Dirichlet Tessellation of Point Pattern*

Description

Computes the Dirichlet tessellation of a spatial point pattern. Also known as the Voronoi or Thiessen tessellation.

Usage

```
dirichlet(X)
```

Arguments

`X`	Spatial point pattern (object of class `"ppp"`).

Details

In a spatial point pattern `X`, the Dirichlet tile associated with a particular point `X[i]` is the region of space that is closer to `X[i]` than to any other point in `X`. The Dirichlet tiles divide the two-dimensional plane into disjoint regions, forming a tessellation.

The Dirichlet tessellation is also known as the Voronoi or Thiessen tessellation.

This function computes the Dirichlet tessellation (within the original window of `X`) using the function `deldir` in the package **deldir**.

To ensure that there is a one-to-one correspondence between the points of `X` and the tiles of `dirichlet(X)`, duplicated points in `X` should first be removed by `X <- unique(X, rule="deldir")`.

The tiles of the tessellation will be computed as polygons if the original window is a rectangle or a polygon. Otherwise the tiles will be computed as binary masks.

Value

A tessellation (object of class `"tess"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`tess`, `delaunay`, `ppp`, `dirichletVertices`

Examples

```
X <- runifpoint(42)
plot(dirichlet(X))
plot(X, add=TRUE)
```

`dirichletAreas` *Compute Areas of Tiles in Dirichlet Tessellation*

Description

Calculates the area of each tile in the Dirichlet-Voronoi tessellation of a point pattern.

Usage

```
dirichletAreas(X)
```

Arguments

`X`	Point pattern (object of class `"ppp"`).

Details

This is an efficient algorithm to calculate the areas of the tiles in the Dirichlet-Voronoi tessellation.

If the window of `X` is a binary pixel mask, the tile areas are computed by counting pixels. Otherwise the areas are computed exactly using analytic geometry.

If any points of `X` are duplicated, the duplicates will have tile area zero.

Value

Numeric vector with one entry for each point of `X`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`dirichlet`, `dirichletVertices`

Examples

```
aa <- dirichletAreas(cells)
```

---

dirichletVertices *Vertices and Edges of Dirichlet Tessellation*

Description

Computes the Dirichlet-Voronoi tessellation of a point pattern and extracts the vertices or edges of the tiles.

Usage

```
dirichletVertices(X)

dirichletEdges(X)
```

Arguments

`X`	Point pattern (object of class "ppp").

Details

These function compute the Dirichlet-Voronoi tessellation of `X` (see `dirichlet`) and extract the vertices or edges of the tiles of the tessellation.

The Dirichlet vertices are the spatial locations which are locally farthest away from `X`, that is, where the distance function of `X` reaches a local maximum.

The Dirichlet edges are the dividing lines equally distant between a pair of points of `X`.

The Dirichlet tessellation of `X` is computed using `dirichlet`. The vertices or edges of all tiles of the tessellation are extracted.

For `dirichletVertices`, any vertex which lies on the boundary of the window of `X` is deleted. The remaining vertices are returned, as a point pattern, without duplicated entries.

Value

`dirichletVertices` returns a point pattern (object of class "ppp") in the same window as `X`.

`dirichletEdges` returns a line segment pattern (object of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

dirichlet, dirichletAreas

Examples

```
plot(dirichlet(cells))

plot(dirichletVertices(cells), add=TRUE)

ed <- dirichletEdges(cells)
```

`dirichletWeights` *Compute Quadrature Weights Based on Dirichlet Tessellation*

Description

Computes quadrature weights for a given set of points, using the areas of tiles in the Dirichlet tessellation.

Usage

```
dirichletWeights(X, window=NULL, exact=TRUE, ...)
```

Arguments

X	Data defining a point pattern.
window	Default window for the point pattern
exact	Logical value. If TRUE, compute exact areas using the package deldir. If FALSE, compute approximate areas using a pixel raster.
...	Ignored.

Details

This function computes a set of quadrature weights for a given pattern of points (typically comprising both "data" and 'dummy" points). See quad.object for an explanation of quadrature weights and quadrature schemes.

The weights are computed using the Dirichlet tessellation. First X and (optionally) window are converted into a point pattern object. Then the Dirichlet tessellation of the points of X is computed. The weight attached to a point of X is the area of its Dirichlet tile (inside the window Window(X)).

If exact=TRUE the Dirichlet tessellation is computed exactly by the Lee-Schachter algorithm using the package deldir. Otherwise a pixel raster approximation is constructed and the areas are approximations to the true weights. In all cases the sum of the weights is equal to the area of the window.

Value

Vector of nonnegative weights for each point in X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quad.object, gridweights

Examples

```
Q <- quadscheme(runifpoispp(10))
X <- as.ppp(Q) # data and dummy points together
w <- dirichletWeights(X, exact=FALSE)
```

---

disc *Circular Window*

Description

Creates a circular window

Usage

```
disc(radius=1, centre=c(0,0), ..., mask=FALSE, npoly=128, delta=NULL)
```

Arguments

radius	Radius of the circle.
centre	The centre of the circle.
mask	Logical flag controlling the type of approximation to a perfect circle. See Details.
npoly	Number of edges of the polygonal approximation, if mask=FALSE. Incompatible with delta.
delta	Tolerance of polygonal approximation: the length of arc that will be replaced by one edge of the polygon. Incompatible with npoly.
...	Arguments passed to as.mask determining the pixel resolution, if mask=TRUE.

Details

This command creates a window object representing a disc, with the given radius and centre.

By default, the circle is approximated by a polygon with npoly edges.

If mask=TRUE, then the disc is approximated by a binary pixel mask. The resolution of the mask is controlled by the arguments ... which are passed to as.mask.

The argument radius must be a single positive number. The argument centre specifies the disc centre: it can be either a numeric vector of length 2 giving the coordinates, or a list(x,y) giving the coordinates of exactly one point, or a point pattern (object of class "ppp") containing exactly one point.

Value

An object of class "owin" (see owin.object) specifying a window.

Note

This function can also be used to generate regular polygons, by setting npoly to a small integer value. For example npoly=5 generates a pentagon and npoly=13 a triskaidecagon.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ellipse, discs, owin.object, owin, as.mask

Examples

```
# unit disc
W <- disc()
# disc of radius 3 centred at x=10, y=5
W <- disc(3, c(10,5))
#
plot(disc())
plot(disc(mask=TRUE))
# nice smooth circle
plot(disc(npoly=256))
# how to control the resolution of the mask
plot(disc(mask=TRUE, dimyx=256))
# check accuracy of approximation
area(disc())/pi
area(disc(mask=TRUE))/pi
```

---

discpartarea *Area of Part of Disc*

Description

Compute area of intersection between a disc and a window

Usage

```
discpartarea(X, r, W=as.owin(X))
```

Arguments

X	Point pattern (object of class "ppp") specifying the centres of the discs. Alternatively, X may be in any format acceptable to as.ppp.
r	Matrix, vector or numeric value specifying the radii of the discs.
W	Window (object of class "owin") with which the discs should be intersected.

Details

This algorithm computes the exact area of the intersection between a window W and a disc (or each of several discs). The centres of the discs are specified by the point pattern X, and their radii are specified by r.

If r is a single numeric value, then the algorithm computes the area of intersection between W and the disc of radius r centred at each point of X, and returns a one-column matrix containing one entry for each point of X.

If r is a vector of length m, then the algorithm returns an n * m matrix in which the entry on row i, column j is the area of the intersection between W and the disc centred at X[i] with radius r[j].

If r is a matrix, it should have one row for each point in X. The algorithm returns a matrix in which the entry on row i, column j is the area of the intersection between W and the disc centred at X[i] with radius r[i,j].

Areas are computed by analytic geometry.

Value

Numeric matrix, with one row for each point of X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin, disc

Examples

```
data(letterR)
X <- runifpoint(3, letterR)
discpartarea(X, 0.2)
```

---

discretise *Safely Convert Point Pattern Window to Binary Mask*

Description

Given a point pattern, discretise its window by converting it to a binary pixel mask, adjusting the mask so that it still contains all the points.

Usage

```
discretise(X, eps = NULL, dimyx = NULL, xy = NULL)
```

Arguments

X	A point pattern (object of class "ppp") to be converted.
eps	(optional) width and height of each pixel
dimyx	(optional) pixel array dimensions
xy	(optional) pixel coordinates

Details

This function modifies the point pattern X by converting its observation window Window(X) to a binary pixel image (a window of type "mask"). It ensures that no points of X are deleted by the discretisation.

The window is first discretised using as.mask. It can happen that points of X that were inside the original window may fall outside the new mask. The discretise function corrects this by augmenting the mask (so that the mask includes any pixel that contains a point of the pattern).

The arguments eps, dimyx and xy control the fineness of the pixel array. They are passed to as.mask.

If eps, dimyx and xy are all absent or NULL, and if the window of X is of type "mask" to start with, then discretise(X) returns X unchanged.

See as.mask for further details about the arguments eps, dimyx, and xy, and the process of converting a window to one of type mask.

Value

A point pattern (object of class "ppp"), identical to X, except that its observation window has been converted to one of type mask.

Error checking

Before doing anything, discretise checks that all the points of the pattern are actually inside the original window. This is guaranteed to be the case if the pattern was constructed using ppp or as.ppp. However anomalies are possible if the point pattern was created or manipulated inappropriately. These will cause an error.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

as.mask

Examples

```
data(demopat)
X <- demopat
plot(X, main="original pattern")
Y <- discretise(X, dimyx=50)
plot(Y, main="discretise(X)")
stopifnot(npoints(X) == npoints(Y))

# what happens if we just convert the window to a mask?
W <- Window(X)
M <- as.mask(W, dimyx=50)
plot(M, main="window of X converted to mask")
plot(X, add=TRUE, pch=16)
plot(X[M], add=TRUE, pch=1, cex=1.5)
XM <- X[M]
cat(paste(npoints(X) - npoints(XM), "points of X lie outside M\n"))
```

discs *Union of Discs*

Description

Make a spatial region composed of discs with given centres and radii.

Usage

```
discs(centres, radii = marks(centres)/2, ...,
      separate = FALSE, mask = FALSE, trim = TRUE,
      delta = NULL, npoly=NULL)
```

Arguments

`centres`	Point pattern giving the locations of centres for the discs.
`radii`	Vector of radii for each disc, or a single number giving a common radius. (Notice that the default assumes that the marks of `X` are *diameters*.)
`...`	Optional arguments passed to `as.mask` to determine the pixel resolution, if `mask=TRUE`.
`separate`	Logical. If `TRUE`, the result is a list containing each disc as a separate entry. If `FALSE` (the default), the result is a window obtained by forming the union of the discs.
`mask`	Logical. If `TRUE`, the result is a binary mask window. If `FALSE`, the result is a polygonal window. Applies only when `separate=FALSE`.
`trim`	Logical value indicating whether to restrict the result to the original window of the `centres`. Applies only when `separate=FALSE`.
`delta`	Argument passed to `disc` to determine the tolerance for the polygonal approximation of each disc. Applies only when `mask=FALSE`. Incompatible with `npoly`.
`npoly`	Argument passed to `disc` to determine the number of edges in the polygonal approximation of each disc. Applies only when `mask=FALSE`. Incompatible with `delta`.

Details

This command is typically applied to a marked point pattern dataset `X` in which the marks represent the sizes of objects. The result is a spatial region representing the space occupied by the objects.

If the marks of `X` represent the diameters of circular objects, then the result of `discs(X)` is a spatial region constructed by taking discs, of the specified diameters, centred at the points of `X`, and forming the union of these discs. If the marks of `X` represent the areas of objects, one could take `discs(X, sqrt(marks(X)/pi))` to produce discs of equivalent area.

A fast algorithm is used to compute the result as a binary mask, when `mask=TRUE`. This option is recommended unless polygons are really necessary.

If `mask=FALSE`, the discs will be constructed as polygons by the function `disc`. To avoid computational problems, by default, the discs will all be constructed using the same physical tolerance value `delta` passed to `disc`. The default is such that the smallest disc will be approximated by a 16-sided polygon. (The argument `npoly` should not normally be used, to avoid computational problems arising with small radii.)

Value

If `separate=FALSE`, a window (object of class `"owin"`).

If `separate=TRUE`, a list of windows.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`disc`, `union.owin`

Examples

```
plot(discs(anemones, mask=TRUE, eps=0.5))
```

`distcdf` *Distribution Function of Interpoint Distance*

Description

Computes the cumulative distribution function of the distance between two independent random points in a given window or windows.

Usage

```
distcdf(W, V=W, ..., dW=1, dV=dW, nr=1024, regularise=TRUE)
```

Arguments

`W`	A window (object of class `"owin"`) containing the first random point.
`V`	Optional. Another window containing the second random point. Defaults to `W`.
`...`	Arguments passed to `as.mask` to determine the pixel resolution for the calculation.
`dV, dW`	Optional. Probability densities (not necessarily normalised) for the first and second random points respectively. Data in any format acceptable to `as.im`, for example, a `function(x,y)` or a pixel image or a numeric value. The default corresponds to a uniform distribution over the window.
`nr`	Integer. The number of values of interpoint distance r for which the CDF will be computed. Should be a large value!
`regularise`	Logical value indicating whether to smooth the results for very small distances, to avoid discretisation artefacts.

Details

This command computes the Cumulative Distribution Function $CDF(r) = Prob(T \leq r)$ of the Euclidean distance $T = \|X_1 - X_2\|$ between two independent random points X_1 and X_2.

In the simplest case, the command `distcdf(W)`, the random points are assumed to be uniformly distributed in the same window `W`.

Alternatively the two random points may be uniformly distributed in two different windows `W` and `V`.

In the most general case the first point X_1 is random in the window `W` with a probability density proportional to `dW`, and the second point X_2 is random in a different window `V` with probability density proportional to `dV`. The values of `dW` and `dV` must be finite and nonnegative.

The calculation is performed by numerical integration of the set covariance function `setcov` for uniformly distributed points, and by computing the covariance function `imcov` in the general case. The accuracy of the result depends on the pixel resolution used to represent the windows: this is controlled by the arguments `...` which are passed to `as.mask`. For example use `eps=0.1` to specify pixels of size 0.1 units.

The arguments `W` or `V` may also be point patterns (objects of class `"ppp"`). The result is the cumulative distribution function of the distance from a randomly selected point in the point pattern, to a randomly selected point in the other point pattern or window.

If `regularise=TRUE` (the default), values of the cumulative distribution function for very short distances are smoothed to avoid discretisation artefacts. Smoothing is applied to all distances shorter than the width of 7 pixels.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`setcov`, `as.mask`.

Examples

```
# The unit disc
B <- disc()
plot(distcdf(B))
```

`distfun` *Distance Map as a Function*

Description

Compute the distance function of an object, and return it as a function.

Usage

```
distfun(X, ...)

  ## S3 method for class 'ppp'
distfun(X, ..., k=1, undef=Inf)

  ## S3 method for class 'psp'
distfun(X, ...)

  ## S3 method for class 'owin'
distfun(X, ..., invert=FALSE)
```

Arguments

X	Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), a window (object of class "owin") or a line segment pattern (object of class "psp").
...	Extra arguments are ignored.
k	An integer. The distance to the kth nearest point will be computed.
undef	The value that should be returned if the distance is undefined (that is, if X contains fewer than k points).
invert	If TRUE, compute the distance transform of the complement of X.

Details

The "distance function" of a set of points A is the mathematical function f such that, for any two-dimensional spatial location (x, y), the function value f(x,y) is the shortest distance from (x, y) to A.

The command `f <- distfun(X)` returns a *function* in the R language, with arguments x,y, that represents the distance function of X. Evaluating the function f in the form `v <- f(x,y)`, where x and y are any numeric vectors of equal length containing coordinates of spatial locations, yields the values of the distance function at these locations. Alternatively x can be a point pattern (object of class "ppp" or "lpp") of locations at which the distance function should be computed (and then y should be missing).

This should be contrasted with the related command distmap which computes the distance function of X on a grid of locations, and returns the distance values in the form of a pixel image.

The result of `f <- distfun(X)` also belongs to the class "funxy" and to the special class "distfun". It can be printed and plotted immediately as shown in the Examples.

A distfun object can be converted to a pixel image using as.im.

Value

A function with arguments x,y. The function also belongs to the class "distfun" which has a method for print. It also belongs to the class "funxy" which has methods for plot, contour and persp.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

distmap, plot.funxy

Examples

```
data(letterR)
f <- distfun(letterR)
f
plot(f)
f(0.2, 0.3)

plot(distfun(letterR, invert=TRUE), eps=0.1)

d <- distfun(cells)
d2 <- distfun(cells, k=2)
d(0.5, 0.5)
d2(0.5, 0.5)

z <- d(japanesepines)
```

---

distfun.lpp — *Distance Map on Linear Network*

Description

Compute the distance function of a point pattern on a linear network.

Usage

```
## S3 method for class 'lpp'
distfun(X, ..., k=1)
```

Arguments

`X`	A point pattern on a linear network (object of class "lpp").
`k`	An integer. The distance to the kth nearest point will be computed.
`...`	Extra arguments are ignored.

Details

On a linear network L, the "geodesic distance function" of a set of points A in L is the mathematical function f such that, for any location s on L, the function value `f(s)` is the shortest-path distance from s to A.

The command `distfun.lpp` is a method for the generic command `distfun` for the class "lpp" of point patterns on a linear network.

If `X` is a point pattern on a linear network, `f <- distfun(X)` returns a *function* in the R language that represents the distance function of `X`. Evaluating the function `f` in the form `v <- f(x,y)`, where `x` and `y` are any numeric vectors of equal length containing coordinates of spatial locations, yields the values of the distance function at these locations. More efficiently `f` can be called in the form `v <- f(x, y, seg, tp)` where `seg` and `tp` are the local coordinates on the network. It can also be called as `v <- f(x)` where `x` is a point pattern on the same linear network.

The function f obtained from f <- distfun(X) also belongs to the class "linfun". It can be printed and plotted immediately as shown in the Examples. It can be converted to a pixel image using as.linim.

Value

A function with arguments x,y and optional arguments seg,tp. It also belongs to the class "linfun" which has methods for plot, print etc.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

linfun, methods.linfun.

To identify *which* point is the nearest neighbour, see nnfun.lpp.

Examples

```
data(letterR)
X <- runiflpp(3, simplenet)
f <- distfun(X)
f
plot(f)

# using a distfun as a covariate in a point process model:
Y <- runiflpp(4, simplenet)
fit <- lppm(Y ~D, covariates=list(D=f))

f(Y)
```

---

distmap *Distance Map*

Description

Compute the distance map of an object, and return it as a pixel image. Generic.

Usage

```
distmap(X, ...)
```

Arguments

X	Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), a window (object of class "owin") or a line segment pattern (object of class "psp").
...	Arguments passed to as.mask to control pixel resolution.

Details

The “distance map” of a set of points A is the function f whose value `f(x)` is defined for any two-dimensional location x as the shortest distance from x to A.

This function computes the distance map of the set `X` and returns the distance map as a pixel image.

This is generic. Methods are provided for point patterns (`distmap.ppp`), line segment patterns (`distmap.psp`) and windows (`distmap.owin`).

Value

A pixel image (object of class `"im"`) whose grey scale values are the values of the distance map.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`distmap.ppp`, `distmap.psp`, `distmap.owin`, `distfun`

Examples

```
  data(cells)
  U <- distmap(cells)
  data(letterR)
  V <- distmap(letterR)
  ## Not run:
  plot(U)
  plot(V)

## End(Not run)
```

`distmap.owin` *Distance Map of Window*

Description

Computes the distance from each pixel to the nearest point in the given window.

Usage

```
  ## S3 method for class 'owin'
distmap(X, ..., discretise=FALSE, invert=FALSE)
```

Arguments

`X`	A window (object of class `"owin"`).
`...`	Arguments passed to `as.mask` to control pixel resolution.
`discretise`	Logical flag controlling the choice of algorithm when `X` is a polygonal window. See Details.
`invert`	If `TRUE`, compute the distance transform of the complement of the window.

Details

The "distance map" of a window W is the function f whose value `f(u)` is defined for any two-dimensional location u as the shortest distance from u to W.

This function computes the distance map of the window `X` and returns the distance map as a pixel image. The greyscale value at a pixel u equals the distance from u to the nearest pixel in `X`.

Additionally, the return value has an attribute `"bdry"` which is also a pixel image. The grey values in `"bdry"` give the distance from each pixel to the bounding rectangle of the image.

If `X` is a binary pixel mask, the distance values computed are not the usual Euclidean distances. Instead the distance between two pixels is measured by the length of the shortest path connecting the two pixels. A path is a series of steps between neighbouring pixels (each pixel has 8 neighbours). This is the standard 'distance transform' algorithm of image processing (Rosenfeld and Kak, 1968; Borgefors, 1986).

If `X` is a polygonal window, then exact Euclidean distances will be computed if `discretise=FALSE`. If `discretise=TRUE` then the window will first be converted to a binary pixel mask and the discrete path distances will be computed.

The arguments `...` are passed to `as.mask` to control the pixel resolution.

This function is a method for the generic `distmap`.

Value

A pixel image (object of class `"im"`) whose greyscale values are the values of the distance map. The return value has an attribute `"bdry"` which is a pixel image.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Borgefors, G. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34** (1986) 344–371.

Rosenfeld, A. and Pfalz, J.L. Distance functions on digital pictures. *Pattern Recognition* **1** (1968) 33-61.

See Also

`distmap`, `distmap.ppp`, `distmap.psp`

Examples

```
  data(letterR)
  U <- distmap(letterR)
  ## Not run:
  plot(U)
  plot(attr(U, "bdry"))

## End(Not run)
```

distmap.ppp *Distance Map of Point Pattern*

Description

Computes the distance from each pixel to the nearest point in the given point pattern.

Usage

```
## S3 method for class 'ppp'
distmap(X, ...)
```

Arguments

X	A point pattern (object of class "ppp").
...	Arguments passed to as.mask to control pixel resolution.

Details

The "distance map" of a point pattern X is the function f whose value f(u) is defined for any two-dimensional location u as the shortest distance from u to X.

This function computes the distance map of the point pattern X and returns the distance map as a pixel image. The greyscale value at a pixel u equals the distance from u to the nearest point of the pattern X.

Additionally, the return value has two attributes, "index" and "bdry", which are also pixel images. The grey values in "bdry" give the distance from each pixel to the bounding rectangle of the image. The grey values in "index" are integers identifying which point of X is closest.

This is a method for the generic function distmap.

Note that this function gives the distance from the *centre of each pixel* to the nearest data point. To compute the exact distance from a given spatial location to the nearest data point in X, use distfun or nncross.

Value

A pixel image (object of class "im") whose greyscale values are the values of the distance map. The return value has attributes "index" and "bdry" which are also pixel images.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Generic function distmap and other methods distmap.psp, distmap.owin.

Generic function distfun.

Nearest neighbour distance nncross

Examples

```
data(cells)
U <- distmap(cells)
## Not run:
plot(U)
plot(attr(U, "bdry"))
plot(attr(U, "index"))

## End(Not run)
```

---

distmap.psp *Distance Map of Line Segment Pattern*

Description

Computes the distance from each pixel to the nearest line segment in the given line segment pattern.

Usage

```
## S3 method for class 'psp'
distmap(X, ...)
```

Arguments

X	A line segment pattern (object of class "psp").
...	Arguments passed to as.mask to control pixel resolution.

Details

The "distance map" of a line segment pattern X is the function f whose value f(u) is defined for any two-dimensional location u as the shortest distance from u to X.

This function computes the distance map of the line segment pattern X and returns the distance map as a pixel image. The greyscale value at a pixel u equals the distance from u to the nearest line segment of the pattern X. Distances are computed using analytic geometry.

Additionally, the return value has two attributes, "index" and "bdry", which are also pixel images. The grey values in "bdry" give the distance from each pixel to the bounding rectangle of the image. The grey values in "index" are integers identifying which line segment of X is closest.

This is a method for the generic function distmap.

Note that this function gives the exact distance from the centre of each pixel to the nearest line segment. To compute the exact distance from the points in a point pattern to the nearest line segment, use distfun or one of the low-level functions nncross or project2segment.

Value

A pixel image (object of class "im") whose greyscale values are the values of the distance map. The return value has attributes "index" and "bdry" which are also pixel images.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

distmap, distmap.owin, distmap.ppp, distfun, nncross, nearestsegment, project2segment.

Examples

```
a <- psp(runif(20),runif(20),runif(20),runif(20), window=owin())
Z <- distmap(a)
plot(Z)
plot(a, add=TRUE)
```

---

divide.linnet   *Divide Linear Network at Cut Points*

Description

Make a tessellation of a linear network by dividing it into pieces demarcated by the points of a point pattern.

Usage

```
divide.linnet(X)
```

Arguments

X	Point pattern on a linear network (object of class "lpp").

Details

The points X are interpreted as dividing the linear network L=as.linnet(X) into separate pieces.

Two locations on L belong to the same piece if and only if they can be joined by a path in L that does not cross any of the points of X.

The result is a tessellation of the network (object of class "lintess") representing the division of L into pieces.

Value

A tessellation on a linear network (object of class "lintess").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> and Greg McSwiggan.

See Also

linnet, lintess.

Examples

```
X <- runiflpp(5, simplenet)
plot(divide.linnet(X))
plot(X, add=TRUE, pch=16)
```

`dkernel` *Kernel distributions and random generation*

Description

Density, distribution function, quantile function and random generation for several distributions used in kernel estimation for numerical data.

Usage

```
dkernel(x, kernel = "gaussian", mean = 0, sd = 1)
pkernel(q, kernel = "gaussian", mean = 0, sd = 1, lower.tail = TRUE)
qkernel(p, kernel = "gaussian", mean = 0, sd = 1, lower.tail = TRUE)
rkernel(n, kernel = "gaussian", mean = 0, sd = 1)
```

Arguments

`x, q`	Vector of quantiles.
`p`	Vector of probabilities.
`kernel`	String name of the kernel. Options are `"gaussian"`, `"rectangular"`, `"triangular"`, `"epanechnikov"`, `"biweight"`, `"cosine"` and `"optcosine"`. (Partial matching is used).
`n`	Number of observations.
`mean`	Mean of distribution.
`sd`	Standard deviation of distribution.
`lower.tail`	logical; if `TRUE` (the default), then probabilities are $P(X \leq x)$, otherwise, $P(X > x)$.

Details

These functions give the probability density, cumulative distribution function, quantile function and random generation for several distributions used in kernel estimation for one-dimensional (numerical) data.

The available kernels are those used in `density.default`, namely `"gaussian"`, `"rectangular"`, `"triangular"`, `"epanechnikov"`, `"biweight"`, `"cosine"` and `"optcosine"`. For more information about these kernels, see `density.default`.

`dkernel` gives the probability density, `pkernel` gives the cumulative distribution function, `qkernel` gives the quantile function, and `rkernel` generates random deviates.

Value

A numeric vector. For `dkernel`, a vector of the same length as `x` containing the corresponding values of the probability density. For `pkernel`, a vector of the same length as `x` containing the corresponding values of the cumulative distribution function. For `qkernel`, a vector of the same length as `p` containing the corresponding quantiles. For `rkernel`, a vector of length `n` containing randomly generated values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>
and Martin Hazelton

See Also

`density.default`, `kernel.factor`

Examples

```
x <- seq(-3,3,length=100)
plot(x, dkernel(x, "epa"), type="l",
         main=c("Epanechnikov kernel", "probability density"))
plot(x, pkernel(x, "opt"), type="l",
         main=c("OptCosine kernel", "cumulative distribution function"))
p <- seq(0,1, length=256)
plot(p, qkernel(p, "biw"), type="l",
         main=c("Biweight kernel", "cumulative distribution function"))
y <- rkernel(100, "tri")
hist(y, main="Random variates from triangular density")
rug(y)
```

`dmixpois` *Mixed Poisson Distribution*

Description

Density, distribution function, quantile function and random generation for a mixture of Poisson distributions.

Usage

```
dmixpois(x, mu, sd, invlink = exp, GHorder = 5)
pmixpois(q, mu, sd, invlink = exp, lower.tail = TRUE, GHorder = 5)
qmixpois(p, mu, sd, invlink = exp, lower.tail = TRUE, GHorder = 5)
rmixpois(n, mu, sd, invlink = exp)
```

Arguments

`x`	vector of (non-negative integer) quantiles.
`q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of random values to return.
`mu`	Mean of the linear predictor. A single numeric value.
`sd`	Standard deviation of the linear predictor. A single numeric value.
`invlink`	Inverse link function. A function in the R language, used to transform the linear predictor into the parameter `lambda` of the Poisson distribution.
`lower.tail`	Logical. If `TRUE` (the default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
`GHorder`	Number of quadrature points in the Gauss-Hermite quadrature approximation. A small positive integer.

Details

These functions are analogous to dpois ppois, qpois and rpois except that they apply to a mixture of Poisson distributions.

In effect, the Poisson mean parameter lambda is randomised by setting lambda = invlink(Z) where Z has a Gaussian $N(\mu, \sigma^2)$ distribution. The default is invlink=exp which means that lambda is lognormal. Set invlink=I to assume that lambda is approximately Normal.

For dmixpois, pmixpois and qmixpois, the probability distribution is approximated using Gauss-Hermite quadrature. For rmixpois, the deviates are simulated exactly.

Value

Numeric vector: dmixpois gives probability masses, ppois gives cumulative probabilities, qpois gives (non-negative integer) quantiles, and rpois generates (non-negative integer) random deviates.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

dpois, gauss.hermite.

Examples

```
dmixpois(7, 10, 1, invlink = I)
dpois(7, 10)

pmixpois(7, log(10), 0.2)
ppois(7, 10)

qmixpois(0.95, log(10), 0.2)
qpois(0.95, 10)

x <- rmixpois(100, log(10), log(1.2))
mean(x)
var(x)
```

---

domain *Extract the Domain of any Spatial Object*

Description

Given a spatial object such as a point pattern, in any number of dimensions, this function extracts the spatial domain in which the object is defined.

Usage

```
domain(X, ...)

## S3 method for class 'ppp'
domain(X, ...)

## S3 method for class 'psp'
domain(X, ...)

## S3 method for class 'im'
domain(X, ...)

## S3 method for class 'ppx'
domain(X, ...)

## S3 method for class 'pp3'
domain(X, ...)

## S3 method for class 'lpp'
domain(X, ...)

## S3 method for class 'ppm'
domain(X, ..., from=c("points", "covariates"))

## S3 method for class 'kppm'
domain(X, ..., from=c("points", "covariates"))

## S3 method for class 'dppm'
domain(X, ..., from=c("points", "covariates"))

## S3 method for class 'lpp'
domain(X, ...)

## S3 method for class 'lppm'
domain(X, ...)

## S3 method for class 'msr'
domain(X, ...)

## S3 method for class 'quad'
domain(X, ...)

## S3 method for class 'quadratcount'
domain(X, ...)

## S3 method for class 'quadrattest'
domain(X, ...)

## S3 method for class 'tess'
domain(X, ...)

## S3 method for class 'linfun'
```

```
domain(X, ...)

 ## S3 method for class 'lintess'
domain(X, ...)

 ## S3 method for class 'im'
domain(X, ...)

 ## S3 method for class 'layered'
domain(X, ...)

 ## S3 method for class 'distfun'
domain(X, ...)

 ## S3 method for class 'nnfun'
domain(X, ...)

 ## S3 method for class 'funxy'
domain(X, ...)

 ## S3 method for class 'rmhmodel'
domain(X, ...)

 ## S3 method for class 'leverage.ppm'
domain(X, ...)

 ## S3 method for class 'influence.ppm'
domain(X, ...)
```

Arguments

X	A spatial object such as a point pattern (in any number of dimensions), line segment pattern or pixel image.
...	Extra arguments. They are ignored by all the methods listed here.
from	Character string. See Details.

Details

The function `domain` is generic.

For a spatial object `X` in any number of dimensions, `domain(X)` extracts the spatial domain in which `X` is defined.

For a two-dimensional object `X`, typically `domain(X)` is the same as `domain(X)`.

The exception is that, if `X` is a point pattern on a linear network (class `"lpp"`) or a point process model on a linear network (class `"lppm"`), then `domain(X)` is the linear network on which the points lie, while `Window(X)` is the two-dimensional window containing the linear network.

The argument `from` applies when `X` is a fitted point process model (object of class `"ppm"`, `"kppm"` or `"dppm"`). If `from="data"` (the default), `domain` extracts the window of the original point pattern data to which the model was fitted. If `from="covariates"` then `domain` returns the window in which the spatial covariates of the model were provided.

Value

A spatial object representing the domain of X. Typically a window (object of class "owin"), a three-dimensional box ("box3"), a multidimensional box ("boxx") or a linear network ("linnet").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

Window, Frame

Examples

```
domain(cells)
domain(bei.extra$elev)
domain(chicago)
```

---

dppapproxkernel   *Approximate Determinantal Point Process Kernel*

Description

Returns an approximation to the kernel of a determinantal point process, as a function of one argument x.

Usage

```
dppapproxkernel(model, trunc = 0.99, W = NULL)
```

Arguments

model	Object of class "detpointprocfamily".
trunc	Numeric specifying how the model truncation is performed. See Details section of simulate.detpointprocfamily.
W	Optional window – undocumented at the moment.

Value

A function

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

dppapproxpcf	*Approximate Pair Correlation Function of Determinantal Point Process Model*

Description

Returns an approximation to the theoretical pair correlation function of a determinantal point process model, as a function of one argument x.

Usage

```
dppapproxpcf(model, trunc = 0.99, W = NULL)
```

Arguments

`model`	Object of class "detpointprocfamily".
`trunc`	Numeric specifying how the model truncation is performed. See Details section of `simulate.detpointprocfamily`.
`W`	Optional window – undocumented at the moment.

Details

This function is usually NOT needed for anything. It only exists for investigative purposes.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
f <- dppapproxpcf(dppMatern(lambda = 100, alpha=.028, nu=1, d=2))
plot(f, xlim = c(0,0.1))
```

dppBessel	*Bessel Type Determinantal Point Process Model*

Description

Function generating an instance of the Bessel-type determinantal point process model.

Usage

```
dppBessel(...)
```

Arguments

`...`	arguments of the form `tag=value` specifying the model parameters. See Details.

Details

The possible parameters are:

- the intensity `lambda` as a positive numeric
- the scale parameter `alpha` as a positive numeric
- the shape parameter `sigma` as a non-negative numeric
- the dimension `d` as a positive integer

Value

An object of class `"detpointprocfamily"`.

Author(s)

Frederic Lavancier and Christophe Biscio. Modified by Ege Rubak `<rubak@math.aau.dk>`, Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`dppCauchy`, `dppGauss`, `dppMatern`, `dppPowerExp`

Examples

```
m <- dppBessel(lambda=100, alpha=.05, sigma=0, d=2)
```

`dppCauchy` *Generalized Cauchy Determinantal Point Process Model*

Description

Function generating an instance of the (generalized) Cauchy determinantal point process model.

Usage

```
dppCauchy(...)
```

Arguments

`...`	arguments of the form `tag=value` specifying the parameters. See Details.

Details

The (generalized) Cauchy DPP is defined in (Lavancier, Møller and Rubak, 2015) The possible parameters are:

- the intensity `lambda` as a positive numeric
- the scale parameter `alpha` as a positive numeric
- the shape parameter `nu` as a positive numeric (artificially required to be less than 20 in the code for numerical stability)
- the dimension `d` as a positive integer

Value

An object of class "detpointprocfamily".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Lavancier, F. Møller, J. and Rubak, E. (2015) Determinantal point process models and statistical inference *Journal of the Royal Statistical Society, Series B* **77**, 853–977.

See Also

dppBessel, dppGauss, dppMatern, dppPowerExp

Examples

```
m <- dppCauchy(lambda=100, alpha=.05, nu=1, d=2)
```

---

dppeigen *Internal function calculating eig and index*

Description

This function is mainly for internal package use and is usually not called by the user.

Usage

```
dppeigen(model, trunc, Wscale, stationary = FALSE)
```

Arguments

model	object of class "detpointprocfamily"
trunc	numeric giving the truncation
Wscale	numeric giving the scale of the window relative to a unit box
stationary	logical indicating whether the stationarity of the model should be used (only works in dimension 2).

Value

A list

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

dppGauss *Gaussian Determinantal Point Process Model*

Description

Function generating an instance of the Gaussian determinantal point process model.

Usage

```
dppGauss(...)
```

Arguments

`...`	arguments of the form `tag=value` specifying the parameters. See Details.

Details

The Gaussian DPP is defined in (Lavancier, Møller and Rubak, 2015) The possible parameters are:

- the intensity `lambda` as a positive numeric
- the scale parameter `alpha` as a positive numeric
- the dimension `d` as a positive integer

Value

An object of class `"detpointprocfamily"`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

Rolf Turner `<r.turner@auckland.ac.nz>`

and Ege Rubak `<rubak@math.aau.dk>`

References

Lavancier, F. Møller, J. and Rubak, E. (2015) Determinantal point process models and statistical inference *Journal of the Royal Statistical Society, Series B* **77**, 853–977.

See Also

`dppBessel`, `dppCauchy`, `dppMatern`, `dppPowerExp`

Examples

```
m <- dppGauss(lambda=100, alpha=.05, d=2)
```

dppkernel	*Extract Kernel from Determinantal Point Process Model Object*

Description

Returns the kernel of a determinantal point process model as a function of one argument `x`.

Usage

```
dppkernel(model, ...)
```

Arguments

`model`	Model of class `"detpointprocfamily"`.
`...`	Arguments passed to `dppapproxkernel` if the exact kernel is unknown

Value

A function

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

Rolf Turner `<r.turner@auckland.ac.nz>`

and Ege Rubak `<rubak@math.aau.dk>`

Examples

```
kernelMatern <- dppkernel(dppMatern(lambda = 100, alpha=.01, nu=1, d=2))
plot(kernelMatern, xlim = c(0,0.1))
```

dppm	*Fit Determinantal Point Process Model*

Description

Fit a determinantal point process model to a point pattern.

Usage

```
  dppm(formula, family, data=NULL,
       ...,
       startpar = NULL,
       method = c("mincon", "clik2", "palm"),
       weightfun=NULL,
       control=list(),
       algorithm="Nelder-Mead",
       statistic="K",
       statargs=list(),
```

```
        rmax = NULL,
        covfunargs=NULL,
        use.gam=FALSE,
        nd=NULL, eps=NULL)
```

Arguments

formula	A formula in the R language specifying the data (on the left side) and the form of the model to be fitted (on the right side). For a stationary model it suffices to provide a point pattern without a formula. See Details.
family	Information specifying the family of point processes to be used in the model. Typically one of the family functions dppGauss, dppMatern, dppCauchy, dppBessel or dppPowerExp. Alternatively a character string giving the name of a family function, or the result of calling one of the family functions. See Details.
data	The values of spatial covariates (other than the Cartesian coordinates) required by the model. A named list of pixel images, functions, windows, tessellations or numeric constants.
...	Additional arguments. See Details.
startpar	Named vector of starting parameter values for the optimization.
method	The fitting method. Either "mincon" for minimum contrast, "clik2" for second order composite likelihood, or "palm" for Palm likelihood. Partially matched.
weightfun	Optional weighting function w in the composite likelihood or Palm likelihood. A function in the R language. See Details.
control	List of control parameters passed to the optimization function optim.
algorithm	Character string determining the mathematical optimisation algorithm to be used by optim. See the argument method of optim.
statistic	Name of the summary statistic to be used for minimum contrast estimation: either "K" or "pcf".
statargs	Optional list of arguments to be used when calculating the statistic. See Details.
rmax	Maximum value of interpoint distance to use in the composite likelihood.
covfunargs,use.gam,nd,eps	Arguments passed to ppm when fitting the intensity.

Details

This function fits a determinantal point process model to a point pattern dataset as described in Lavancier et al. (2015).

The model to be fitted is specified by the arguments formula and family.

The argument formula should normally be a formula in the R language. The left hand side of the formula specifies the point pattern dataset to which the model should be fitted. This should be a single argument which may be a point pattern (object of class "ppp") or a quadrature scheme (object of class "quad"). The right hand side of the formula is called the trend and specifies the form of the *logarithm of the intensity* of the process. Alternatively the argument formula may be a point pattern or quadrature scheme, and the trend formula is taken to be ~1.

The argument family specifies the family of point processes to be used in the model. It is typically one of the family functions dppGauss, dppMatern, dppCauchy, dppBessel or dppPowerExp. Alternatively it may be a character string giving the name of a family function, or the result of calling one of the family functions. A family function belongs to class "detpointprocfamilyfun". The result of calling a family function is a point process family, which belongs to class "detpointprocfamily".

The algorithm first estimates the intensity function of the point process using ppm. If the trend formula is ~1 (the default if a point pattern or quadrature scheme is given rather than a "formula") then the model is *homogeneous*. The algorithm begins by estimating the intensity as the number of points divided by the area of the window. Otherwise, the model is *inhomogeneous*. The algorithm begins by fitting a Poisson process with log intensity of the form specified by the formula trend. (See ppm for further explanation).

The interaction parameters of the model are then fitted either by minimum contrast estimation, or by maximum composite likelihood.

Minimum contrast: If method = "mincon" (the default) interaction parameters of the model will be fitted by minimum contrast estimation, that is, by matching the theoretical K-function of the model to the empirical K-function of the data, as explained in mincontrast.

For a homogeneous model (trend = ~1) the empirical K-function of the data is computed using Kest, and the interaction parameters of the model are estimated by the method of minimum contrast.

For an inhomogeneous model, the inhomogeneous K function is estimated by Kinhom using the fitted intensity. Then the interaction parameters of the model are estimated by the method of minimum contrast using the inhomogeneous K function. This two-step estimation procedure is heavily inspired by Waagepetersen (2007).

If statistic="pcf" then instead of using the K-function, the algorithm will use the pair correlation function pcf for homogeneous models and the inhomogeneous pair correlation function pcfinhom for inhomogeneous models. In this case, the smoothing parameters of the pair correlation can be controlled using the argument statargs, as shown in the Examples.

Additional arguments ... will be passed to mincontrast to control the minimum contrast fitting algorithm.

Composite likelihood: If method = "clik2" the interaction parameters of the model will be fitted by maximising the second-order composite likelihood (Guan, 2006). The log composite likelihood is

$$\sum_{i,j} w(d_{ij}) \log \rho(d_{ij};\theta) - \left(\sum_{i,j} w(d_{ij}) \right) \log \int_D \int_D w(\|u-v\|)\rho(\|u-v\|;\theta)\, du\, dv$$

where the sums are taken over all pairs of data points x_i, x_j separated by a distance $d_{ij} = \|x_i - x_j\|$ less than rmax, and the double integral is taken over all pairs of locations u, v in the spatial window of the data. Here $\rho(d;\theta)$ is the pair correlation function of the model with cluster parameters θ.

The function w in the composite likelihood is a weighting function and may be chosen arbitrarily. It is specified by the argument weightfun. If this is missing or NULL then the default is a threshold weight function, $w(d) = 1(d \le R)$, where R is rmax/2.

Palm likelihood: If method = "palm" the interaction parameters of the model will be fitted by maximising the Palm loglikelihood (Tanaka et al, 2008)

$$\sum_{i,j} w(x_i, x_j) \log \lambda_P(x_j \mid x_i; \theta) - \int_D w(x_i, u)\lambda_P(u \mid x_i; \theta) \mathrm{d}u$$

with the same notation as above. Here $\lambda_P(u|v;\theta$ is the Palm intensity of the model at location u given there is a point at v.

In all three methods, the optimisation is performed by the generic optimisation algorithm optim. The behaviour of this algorithm can be modified using the argument control. Useful control arguments include trace, maxit and abstol (documented in the help for optim).

Finally, it is also possible to fix any parameters desired before the optimisation by specifying them as name=value in the call to the family function. See Examples.

Value

An object of class "dppm" representing the fitted model. There are methods for printing, plotting, predicting and simulating objects of this class.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Lavancier, F. Møller, J. and Rubak, E. (2015) Determinantal point process models and statistical inference *Journal of the Royal Statistical Society, Series B* **77**, 853–977.

Guan, Y. (2006) A composite likelihood approach in fitting spatial point process models. *Journal of the American Statistical Association* **101**, 1502–1512.

Tanaka, U. and Ogata, Y. and Stoyan, D. (2008) Parameter estimation and model selection for Neyman-Scott point processes. *Biometrical Journal* **50**, 43–57.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

methods for dppm objects: plot.dppm, fitted.dppm, predict.dppm, simulate.dppm, methods.dppm, as.ppm.dppm, Kmodel.dppm, pcfmodel.dppm.

Minimum contrast fitting algorithm: mincontrast.

Deterimantal point process models: dppGauss, dppMatern, dppCauchy, dppBessel, dppPowerExp,

Summary statistics: Kest, Kinhom, pcf, pcfinhom.

See also ppm

Examples

```
jpines <- residualspaper$Fig1


dppm(jpines ~ 1, dppGauss)

dppm(jpines ~ 1, dppGauss, method="c")
dppm(jpines ~ 1, dppGauss, method="p")

# Fixing the intensity to lambda=2 rather than the Poisson MLE 2.04:
dppm(jpines ~ 1, dppGauss(lambda=2))

if(interactive()) {
 # The following is quite slow (using K-function)
 dppm(jpines ~ x, dppMatern)
```

```
}

 # much faster using pair correlation function
dppm(jpines ~ x, dppMatern, statistic="pcf", statargs=list(stoyan=0.2))

# Fixing the Matern shape parameter to nu=2 rather than estimating it:
dppm(jpines ~ x, dppMatern(nu=2))
```

---

dppMatern *Whittle-Matern Determinantal Point Process Model*

Description

Function generating an instance of the Whittle-Matern determinantal point process model

Usage

```
dppMatern(...)
```

Arguments

`...`	arguments of the form `tag=value` specifying the parameters. See Details.

Details

The Whittle-Matérn DPP is defined in (Lavancier, Møller and Rubak, 2015) The possible parameters are:

- the intensity `lambda` as a positive numeric
- the scale parameter `alpha` as a positive numeric
- the shape parameter `nu` as a positive numeric (artificially required to be less than 20 in the code for numerical stability)
- the dimension `d` as a positive integer

Value

An object of class `"detpointprocfamily"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Lavancier, F. Møller, J. and Rubak, E. (2015) Determinantal point process models and statistical inference *Journal of the Royal Statistical Society, Series B* **77**, 853–977.

See Also

`dppBessel`, `dppCauchy`, `dppGauss`, `dppPowerExp`

Examples

```
m <- dppMatern(lambda=100, alpha=.02, nu=1, d=2)
```

`dppparbounds` *Parameter Bound for a Determinantal Point Process Model*

Description

Returns the lower and upper bound for a specific parameter of a determinantal point process model when all other parameters are fixed.

Usage

```
dppparbounds(model, name, ...)
```

Arguments

`model`	Model of class `"detpointprocfamily"`.
`name`	name of the parameter for which the bound should be computed.
`...`	Additional arguments passed to the `parbounds` function of the given model

Value

A `data.frame` containing lower and upper bounds.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

Rolf Turner <`r.turner@auckland.ac.nz`>

and Ege Rubak <`rubak@math.aau.dk`>

Examples

```
model <- dppMatern(lambda=100, alpha=.01, nu=1, d=2)
dppparbounds(model, "lambda")
```

`dppPowerExp` *Power Exponential Spectral Determinantal Point Process Model*

Description

Function generating an instance of the Power Exponential Spectral determinantal point process model.

Usage

```
dppPowerExp(...)
```

Arguments

`...` arguments of the form `tag=value` specifying the parameters. See Details.

Details

The Power Exponential Spectral DPP is defined in (Lavancier, Møller and Rubak, 2015) The possible parameters are:

- the intensity `lambda` as a positive numeric
- the scale parameter `alpha` as a positive numeric
- the shape parameter `nu` as a positive numeric (artificially required to be less than 20 in the code for numerical stability)
- the dimension `d` as a positive integer

Value

An object of class `"detpointprocfamily"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Lavancier, F. Møller, J. and Rubak, E. (2015) Determinantal point process models and statistical inference *Journal of the Royal Statistical Society, Series B* **77**, 853–977.

See Also

`dppBessel`, `dppCauchy`, `dppGauss`, `dppMatern`

Examples

```
m <- dppPowerExp(lambda=100, alpha=.01, nu=1, d=2)
```

---

dppspecden — *Extract Spectral Density from Determinantal Point Process Model Object*

Description

Returns the spectral density of a determinantal point process model as a function of one argument `x`.

Usage

```
dppspecden(model)
```

Arguments

model	Model of class "detpointprocfamily".

Value

A function

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

dppspecdenrange

Examples

```
model <- dppMatern(lambda = 100, alpha=.01, nu=1, d=2)
dppspecden(model)
```

---

dppspecdenrange *Range of Spectral Density of a Determinantal Point Process Model*

Description

Computes the range of the spectral density of a determinantal point process model.

Usage

```
dppspecdenrange(model)
```

Arguments

model	Model of class "detpointprocfamily".

Value

Numeric value (possibly Inf).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

dppspecden

Examples

```
m <- dppBessel(lambda=100, alpha=0.05, sigma=1, d=2)
dppspecdenrange(m)
```

`dummify` *Convert Data to Numeric Values by Constructing Dummy Variables*

Description

Converts data of any kind to numeric values. A factor is expanded to a set of dummy variables.

Usage

```
dummify(x)
```

Arguments

`x`	Vector, factor, matrix or data frame to be converted.

Details

This function converts data (such as a factor) to numeric values in order that the user may calculate, for example, the mean, variance, covariance and correlation of the data.

If `x` is a numeric vector or integer vector, it is returned unchanged.

If `x` is a logical vector, it is converted to a 0-1 matrix with 2 columns. The first column contains a 1 if the logical value is `FALSE`, and the second column contains a 1 if the logical value is `TRUE`.

If `x` is a complex vector, it is converted to a matrix with 2 columns, containing the real and imaginary parts.

If `x` is a factor, the result is a matrix of 0-1 dummy variables. The matrix has one column for each possible level of the factor. The `(i,j)` entry is equal to 1 when the `i`th factor value equals the `j`th level, and is equal to 0 otherwise.

If `x` is a matrix or data frame, the appropriate conversion is applied to each column of `x`.

Note that, unlike `model.matrix`, this command converts a factor into a full set of dummy variables (one column for each level of the factor).

Value

A numeric matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Examples

```
chara <- sample(letters[1:3], 8, replace=TRUE)
logi <- (runif(8) < 0.3)
comp <- round(4*runif(8) + 3*runif(8) * 1i, 1)
nume <- 8:1 + 0.1
df <- data.frame(nume, chara, logi, comp)
df
dummify(df)
```

dummy.ppm — *Extract Dummy Points Used to Fit a Point Process Model*

Description

Given a fitted point process model, this function extracts the 'dummy points' of the quadrature scheme used to fit the model.

Usage

```
dummy.ppm(object, drop=FALSE)
```

Arguments

`object`	fitted point process model (an object of class "ppm").
`drop`	Logical value determining whether to delete dummy points that were not used to fit the model.

Details

An object of class "ppm" represents a point process model that has been fitted to data. It is typically produced by the model-fitting algorithm `ppm`.

The maximum pseudolikelihood algorithm in `ppm` approximates the pseudolikelihood integral by a sum over a finite set of quadrature points, which is constructed by augmenting the original data point pattern by a set of "dummy" points. The fitted model object returned by `ppm` contains complete information about this quadrature scheme. See `ppm` or `ppm.object` for further information.

This function `dummy.ppm` extracts the dummy points of the quadrature scheme. A typical use of this function would be to count the number of dummy points, to gauge the accuracy of the approximation to the exact pseudolikelihood.

It may happen that some dummy points are not actually used in fitting the model (typically because the value of a covariate is `NA` at these points). The argument `drop` specifies whether these unused dummy points shall be deleted (`drop=TRUE`) or retained (`drop=FALSE`) in the return value.

See `ppm.object` for a list of all operations that can be performed on objects of class "ppm".

Value

A point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppm.object`, `ppp.object`, `ppm`

Examples

```
data(cells)
fit <- ppm(cells, ~1, Strauss(r=0.1))
X <- dummy.ppm(fit)
npoints(X)
# this is the number of dummy points in the quadrature scheme
```

`duplicated.ppp` *Determine Duplicated Points in a Spatial Point Pattern*

Description

Determines which points in a spatial point pattern are duplicates of previous points, and returns a logical vector.

Usage

```
 ## S3 method for class 'ppp'
duplicated(x, ..., rule=c("spatstat", "deldir", "unmark"))

 ## S3 method for class 'ppx'
duplicated(x, ...)

 ## S3 method for class 'ppp'
anyDuplicated(x, ...)

 ## S3 method for class 'ppx'
anyDuplicated(x, ...)
```

Arguments

`x`	A spatial point pattern (object of class "ppp" or "ppx").
`...`	Ignored.
`rule`	Character string. The rule for determining duplicated points.

Details

These are methods for the generic functions `duplicated` and `anyDuplicated` for point pattern datasets (of class "ppp", see `ppp.object`, or class "ppx").

`anyDuplicated(x)` is a faster version of `any(duplicated(x))`.

Two points in a point pattern are deemed to be identical if their x, y coordinates are the same, and their marks are also the same (if they carry marks). The Examples section illustrates how it is possible for a point pattern to contain a pair of identical points.

This function determines which points in `x` duplicate other points that appeared earlier in the sequence. It returns a logical vector with entries that are `TRUE` for duplicated points and `FALSE` for unique (non-duplicated) points.

If `rule="spatstat"` (the default), two points are deemed identical if their coordinates are equal according to `==`, *and* their marks are equal according to `==`. This is the most stringent possible test. If `rule="unmark"`, duplicated points are determined by testing equality of their coordinates only, using ==. If rule="deldir", duplicated points are determined by testing equality of their coordinates only, using the function duplicatedxy in the package **deldir**, which currently uses duplicated.data.frame. Setting rule="deldir" will ensure consistency with functions in the **deldir** package.

Value

duplicated(x) returns a logical vector of length equal to the number of points in x.

anyDuplicated(x) is a number equal to 0 if there are no duplicated points, and otherwise is equal to the index of the first duplicated point.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppp.object, unique.ppp, multiplicity.ppp

Examples

```
X <- ppp(c(1,1,0.5), c(2,2,1), window=square(3))
duplicated(X)
duplicated(X, rule="deldir")
```

---

edge.Ripley *Ripley's Isotropic Edge Correction*

Description

Computes Ripley's isotropic edge correction weights for a point pattern.

Usage

```
edge.Ripley(X, r, W = Window(X), method = "C", maxweight = 100)

rmax.Ripley(W)
```

Arguments

X	Point pattern (object of class "ppp").
W	Window for which the edge correction is required.
r	Vector or matrix of interpoint distances for which the edge correction should be computed.
method	Choice of algorithm. Either "interpreted" or "C". This is needed only for debugging purposes.
maxweight	Maximum permitted value of the edge correction weight.

Details

The function `edge.Ripley` computes Ripley's (1977) isotropic edge correction weight, which is used in estimating the K function and in many other contexts.

The function `rmax.Ripley` computes the maximum value of distance r for which the isotropic edge correction estimate of $K(r)$ is valid.

For a single point x in a window W, and a distance $r > 0$, the isotropic edge correction weight is

$$e(u, r) = \frac{2\pi r}{\text{length}(c(u, r) \cap W)}$$

where $c(u, r)$ is the circle of radius r centred at the point u. The denominator is the length of the overlap between this circle and the window W.

The function `edge.Ripley` computes this edge correction weight for each point in the point pattern `X` and for each corresponding distance value in the vector or matrix `r`.

If `r` is a vector, with one entry for each point in `X`, then the result is a vector containing the edge correction weights `e(X[i], r[i])` for each `i`.

If `r` is a matrix, with one row for each point in `X`, then the result is a matrix whose `i,j` entry gives the edge correction weight `e(X[i], r[i,j])`. For example `edge.Ripley(X, pairdist(X))` computes all the edge corrections required for the K-function.

If any value of the edge correction weight exceeds `maxwt`, it is set to `maxwt`.

The function `rmax.Ripley` computes the smallest distance r such that it is possible to draw a circle of radius r, centred at a point of `W`, such that the circle does not intersect the interior of `W`.

Value

A numeric vector or matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

`edge.Trans`, `rmax.Trans`, `Kest`

Examples

```
v <- edge.Ripley(cells, pairdist(cells))

rmax.Ripley(Window(cells))
```

edge.Trans *Translation Edge Correction*

Description

Computes Ohser and Stoyan's translation edge correction weights for a point pattern.

Usage

```
edge.Trans(X, Y = X, W = Window(X),
      exact = FALSE, paired = FALSE,
      ...,
      trim = spatstat.options("maxedgewt"),
      dx=NULL, dy=NULL,
      give.rmax=FALSE, gW=NULL)

rmax.Trans(W, g=setcov(W))
```

Arguments

X,Y	Point patterns (objects of class "ppp").
W	Window for which the edge correction is required.
exact	Logical. If TRUE, a slow algorithm will be used to compute the exact value. If FALSE, a fast algorithm will be used to compute the approximate value.
paired	Logical value indicating whether X and Y are paired. If TRUE, compute the edge correction for corresponding points X[i], Y[i] for all i. If FALSE, compute the edge correction for each possible pair of points X[i], Y[j] for all i and j.
...	Ignored.
trim	Maximum permitted value of the edge correction weight.
dx,dy	Alternative data giving the x and y coordinates of the vector differences between the points. Incompatible with X and Y. See Details.
give.rmax	Logical. If TRUE, also compute the value of rmax.Trans(W) and return it as an attribute of the result.
g, gW	Optional. Set covariance of W, if it has already been computed. Not required if W is a rectangle.

Details

The function edge.Trans computes Ohser and Stoyan's translation edge correction weight, which is used in estimating the K function and in many other contexts.

The function rmax.Trans computes the maximum value of distance r for which the translation edge correction estimate of $K(r)$ is valid.

For a pair of points x and y in a window W, the translation edge correction weight is

$$e(u,r) = \frac{\text{area}(W)}{\text{area}(W \cap (W + y - x))}$$

where $W + y - x$ is the result of shifting the window W by the vector $y - x$. The denominator is the area of the overlap between this shifted window and the original window.

The function edge.Trans computes this edge correction weight. If paired=TRUE, then X and Y should contain the same number of points. The result is a vector containing the edge correction weights e(X[i], Y[i]) for each i.

If paired=FALSE, then the result is a matrix whose i,j entry gives the edge correction weight e(X[i], Y[j]).

Computation is exact if the window is a rectangle. Otherwise,

- if exact=TRUE, the edge correction weights are computed exactly using overlap.owin, which can be quite slow.
- if exact=FALSE (the default), the weights are computed rapidly by evaluating the set covariance function setcov using the Fast Fourier Transform.

If any value of the edge correction weight exceeds trim, it is set to trim.

The arguments dx and dy can be provided as an alternative to X and Y. If paired=TRUE then dx,dy should be vectors of equal length such that the vector difference of the ith pair is c(dx[i], dy[i]). If paired=FALSE then dx,dy should be matrices of the same dimensions, such that the vector difference between X[i] and Y[j] is c(dx[i,j], dy[i,j]). The argument W is needed.

The value of rmax.Trans is the shortest distance from the origin $(0,0)$ to the boundary of the support of the set covariance function of W. It is computed by pixel approximation using setcov, unless W is a rectangle, when rmax.Trans(W) is the length of the shortest side of the rectangle.

Value

Numeric vector or matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

References

Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.

See Also

rmax.Trans, edge.Ripley, setcov, Kest

Examples

```
v <- edge.Trans(cells)
rmax.Trans(Window(cells))
```

edges *Extract Boundary Edges of a Window.*

Description

Extracts the boundary edges of a window and returns them as a line segment pattern.

Usage

```
edges(x, ..., window = NULL, check = FALSE)
```

Arguments

x	A window (object of class "owin"), or data acceptable to as.owin, specifying the window whose boundary is to be extracted.
...	Ignored.
window	Window to contain the resulting line segments. Defaults to as.rectangle(x).
check	Logical. Whether to check the validity of the resulting segment pattern.

Details

The boundary edges of the window x will be extracted as a line segment pattern.

Value

A line segment pattern (object of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

perimeter for calculating the total length of the boundary.

Examples

```
edges(square(1))
edges(letterR)
```

`edges2triangles` *List Triangles in a Graph*

Description

Given a list of edges between vertices, compile a list of all triangles formed by these edges.

Usage

```
edges2triangles(iedge, jedge, nvert=max(iedge, jedge), ...,
                check=TRUE, friendly=rep(TRUE, nvert))
```

Arguments

`iedge,jedge`	Integer vectors, of equal length, specifying the edges.
`nvert`	Number of vertices in the network.
`...`	Ignored
`check`	Logical. Whether to check validity of input data.
`friendly`	Optional. For advanced use. See Details.

Details

This low level function finds all the triangles (cliques of size 3) in a finite graph with `nvert` vertices and with edges specified by `iedge, jedge`.

The interpretation of `iedge, jedge` is that each successive pair of entries specifies an edge in the graph. The kth edge joins vertex `iedge[k]` to vertex `jedge[k]`. Entries of `iedge` and `jedge` must be integers from 1 to `nvert`.

To improve efficiency in some applications, the optional argument `friendly` can be used. It should be a logical vector of length `nvert` specifying a labelling of the vertices, such that two vertices `j,k` which are *not* friendly (`friendly[j] = friendly[k] = FALSE`) are *never* connected by an edge.

Value

A 3-column matrix of integers, in which each row represents a triangle.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`edges2vees`

Examples

```
i <- c(1, 2, 5, 5, 1, 4, 2)
j <- c(2, 3, 3, 1, 3, 2, 5)
edges2triangles(i, j)
```

edges2vees *List Dihedral Triples in a Graph*

Description

Given a list of edges between vertices, compile a list of all 'vees' or dihedral triples formed by these edges.

Usage

```
edges2vees(iedge, jedge, nvert=max(iedge, jedge), ...,
                    check=TRUE)
```

Arguments

iedge,jedge	Integer vectors, of equal length, specifying the edges.
nvert	Number of vertices in the network.
...	Ignored
check	Logical. Whether to check validity of input data.

Details

Given a finite graph with nvert vertices and with edges specified by iedge, jedge, this low-level function finds all 'vees' or 'dihedral triples' in the graph, that is, all triples of vertices (i,j,k) where i and j are joined by an edge and i and k are joined by an edge.

The interpretation of iedge, jedge is that each successive pair of entries specifies an edge in the graph. The kth edge joins vertex iedge[k] to vertex jedge[k]. Entries of iedge and jedge must be integers from 1 to nvert.

Value

A 3-column matrix of integers, in which each row represents a triple of vertices, with the first vertex joined to the other two vertices.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

edges2triangles

Examples

```
i <- c(1, 2, 5, 5, 1, 4, 2)
j <- c(2, 3, 3, 1, 3, 2, 5)
edges2vees(i, j)
```

`edit.hyperframe` *Invoke Text Editor on Hyperframe*

Description

Invokes a text editor allowing the user to inspect and change entries in a hyperframe.

Usage

```
## S3 method for class 'hyperframe'
edit(name, ...)
```

Arguments

`name`	A hyperframe (object of class `"hyperframe"`).
`...`	Other arguments passed to `edit.data.frame`.

Details

The function `edit` is generic. This function is the methods for objects of class `"hyperframe"`.

The hyperframe `name` is converted to a data frame or array, and the text editor is invoked. The user can change entries in the columns of data, and create new columns of data.

Only the columns of atomic data (numbers, characters, factor values etc) can be edited.

Note that the original object `name` is not changed; the function returns the edited dataset.

Value

Another hyperframe.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`edit.data.frame`, `edit.ppp`

Examples

```
if(interactive()) Z <- edit(flu)
```

edit.ppp *Invoke Text Editor on Spatial Data*

Description

Invokes a text editor allowing the user to inspect and change entries in a spatial dataset.

Usage

```
## S3 method for class 'ppp'
edit(name, ...)

## S3 method for class 'psp'
edit(name, ...)

## S3 method for class 'im'
edit(name, ...)
```

Arguments

`name`	A spatial dataset (object of class "ppp", "psp" or "im").
`...`	Other arguments passed to `edit.data.frame`.

Details

The function `edit` is generic. These functions are methods for spatial objects of class "ppp", "psp" and "im".

The spatial dataset `name` is converted to a data frame or array, and the text editor is invoked. The user can change the values of spatial coordinates or marks of the points in a point pattern, or the coordinates or marks of the segments in a segment pattern, or the pixel values in an image. The names of the columns of marks can also be edited.

If `name` is a pixel image, it is converted to a matrix and displayed in the same spatial orientation as if the image had been plotted.

Note that the original object `name` is not changed; the function returns the edited dataset.

Value

Object of the same kind as `name` containing the edited data.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`edit.data.frame`, `edit.hyperframe`

Examples

```
if(interactive()) Z <- edit(cells)
```

---

eem *Exponential Energy Marks*

Description

Given a point process model fitted to a point pattern, compute the Stoyan-Grabarnik diagnostic "exponential energy marks" for the data points.

Usage

```
eem(fit, check=TRUE)
```

Arguments

`fit`	The fitted point process model. An object of class `"ppm"`.
`check`	Logical value indicating whether to check the internal format of `fit`. If there is any possibility that this object has been restored from a dump file, or has otherwise lost track of the environment where it was originally computed, set `check=TRUE`.

Details

Stoyan and Grabarnik (1991) proposed a diagnostic tool for point process models fitted to spatial point pattern data. Each point x_i of the data pattern X is given a 'mark' or 'weight'

$$m_i = \frac{1}{\hat{\lambda}(x_i, X)}$$

where $\hat{\lambda}(x_i, X)$ is the conditional intensity of the fitted model. If the fitted model is correct, then the sum of these marks for all points in a region B has expected value equal to the area of B.

The argument `fit` must be a fitted point process model (object of class `"ppm"`). Such objects are produced by the maximum pseudolikelihood fitting algorithm `ppm`). This fitted model object contains complete information about the original data pattern and the model that was fitted to it.

The value returned by `eem` is the vector of weights $m[i]$ associated with the points $x[i]$ of the original data pattern. The original data pattern (in corresponding order) can be extracted from `fit` using `data.ppm`.

The function `diagnose.ppm` produces a set of sensible diagnostic plots based on these weights.

Value

A vector containing the values of the exponential energy mark for each point in the pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Stoyan, D. and Grabarnik, P. (1991) Second-order characteristics for stochastic structures connected with Gibbs point processes. *Mathematische Nachrichten*, 151:95–100.

See Also

diagnose.ppm, ppm.object, data.ppm, residuals.ppm, ppm

Examples

```
data(cells)
fit <- ppm(cells, ~x, Strauss(r=0.15))
ee <- eem(fit)
sum(ee)/area(Window(cells)) # should be about 1 if model is correct
Y <- setmarks(cells, ee)
plot(Y, main="Cells data\n Exponential energy marks")
```

---

effectfun *Compute Fitted Effect of a Spatial Covariate in a Point Process Model*

Description

Compute the trend or intensity of a fitted point process model as a function of one of its covariates.

Usage

```
effectfun(model, covname, ..., se.fit=FALSE)
```

Arguments

model	A fitted point process model (object of class "ppm", "kppm", "lppm", "dppm", "rppm" or "profilepl").
covname	The name of the covariate. A character string. (Needed only if the model has more than one covariate.)
...	The fixed values of other covariates (in the form name=value) if required.
se.fit	Logical. If TRUE, asymptotic standard errors of the estimates will be computed, together with a 95% confidence interval.

Details

The object model should be an object of class "ppm", "kppm", "lppm", "dppm", "rppm" or "profilepl" representing a point process model fitted to point pattern data.

The model's trend formula should involve a spatial covariate named covname. This could be "x" or "y" representing one of the Cartesian coordinates. More commonly the covariate is another, external variable that was supplied when fitting the model.

The command effectfun computes the fitted trend of the point process model as a function of the covariate named covname. The return value can be plotted immediately, giving a plot of the fitted trend against the value of the covariate.

If the model also involves covariates other than covname, then these covariates will be held fixed. Values for these other covariates must be provided as arguments to effectfun in the form name=value.

If se.fit=TRUE, the algorithm also calculates the asymptotic standard error of the fitted trend, and a (pointwise) asymptotic 95% confidence interval for the true trend.

This command is just a wrapper for the prediction method predict.ppm. For more complicated computations about the fitted intensity, use predict.ppm.

Value

A data frame containing a column of values of the covariate and a column of values of the fitted trend. If se.fit=TRUE, there are 3 additional columns containing the standard error and the upper and lower limits of a confidence interval.

If the covariate named covname is numeric (rather than a factor or logical variable), the return value is also of class "fv" so that it can be plotted immediately.

Trend and intensity

For a Poisson point process model, the trend is the same as the intensity of the point process. For a more general Gibbs model, the trend is the first order potential in the model (the first order term in the Gibbs representation). In Poisson or Gibbs models fitted by ppm, the trend is the only part of the model that depends on the covariates.

Determinantal point process models with fixed intensity

The function dppm which fits a determinantal point process model allows the user to specify the intensity lambda. In such cases the effect function is undefined, and effectfun stops with an error message.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

See Also

ppm, predict.ppm, fv.object

Examples

```
X <- copper$SouthPoints
D <- distfun(copper$SouthLines)
fit <- ppm(X ~ polynom(D, 5))
effectfun(fit)
plot(effectfun(fit, se.fit=TRUE))

fitx <- ppm(X ~ x + polynom(D, 5))
plot(effectfun(fitx, "D", x=20))
```

ellipse *Elliptical Window.*

Description

Create an elliptical window.

Usage

```
ellipse(a, b, centre=c(0,0), phi=0, ..., mask=FALSE, npoly = 128)
```

Arguments

a,b	The half-lengths of the axes of the ellipse.
centre	The centre of the ellipse.
phi	The (anti-clockwise) angle through which the ellipse should be rotated (about its centre) starting from an orientation in which the axis of half-length a is horizontal.
mask	Logical value controlling the type of approximation to a perfect ellipse. See Details.
...	Arguments passed to as.mask to determine the pixel resolution, if mask is TRUE.
npoly	The number of edges in the polygonal approximation to the ellipse.

Details

This command creates a window object representing an ellipse with the given centre and axes.

By default, the ellipse is approximated by a polygon with npoly edges.

If mask=TRUE, then the ellipse is approximated by a binary pixel mask. The resolution of the mask is controlled by the arguments ... which are passed to as.mask.

The arguments a and b must be single positive numbers. The argument centre specifies the ellipse centre: it can be either a numeric vector of length 2 giving the coordinates, or a list(x,y) giving the coordinates of exactly one point, or a point pattern (object of class "ppp") containing exactly one point.

Value

An object of class owin (either of type "polygonal" or of type "mask") specifying an elliptical window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

disc, owin.object, owin, as.mask

Examples

```
W <- ellipse(a=5,b=2,centre=c(5,1),phi=pi/6)
plot(W,lwd=2,border="red")
WM <- ellipse(a=5,b=2,centre=c(5,1),phi=pi/6,mask=TRUE,dimyx=512)
plot(WM,add=TRUE,box=FALSE)
```

Emark *Diagnostics for random marking*

Description

Estimate the summary functions $E(r)$ and $V(r)$ for a marked point pattern, proposed by Schlather et al (2004) as diagnostics for dependence between the points and the marks.

Usage

```
Emark(X, r=NULL,
         correction=c("isotropic", "Ripley", "translate"),
         method="density", ..., normalise=FALSE)
Vmark(X, r=NULL,
         correction=c("isotropic", "Ripley", "translate"),
         method="density", ..., normalise=FALSE)
```

Arguments

`X`	The observed point pattern. An object of class "ppp" or something acceptable to `as.ppp`. The pattern should have numeric marks.
`r`	Optional. Numeric vector. The values of the argument r at which the function $E(r)$ or $V(r)$ should be evaluated. There is a sensible default.
`correction`	A character vector containing any selection of the options "isotropic", "Ripley" or "translate". It specifies the edge correction(s) to be applied.
`method`	A character vector indicating the user's choice of density estimation technique to be used. Options are "density", "loess", "sm" and "smrep".
`...`	Arguments passed to the density estimation routine (`density`, `loess` or `sm.density`) selected by `method`.
`normalise`	If`TRUE`, normalise the estimate of $E(r)$ or $V(r)$ so that it would have value equal to 1 if the marks are independent of the points.

Details

For a marked point process, Schlather et al (2004) defined the functions $E(r)$ and $V(r)$ to be the conditional mean and conditional variance of the mark attached to a typical random point, given that there exists another random point at a distance r away from it.

More formally,

$$E(r) = E_{0u}[M(0)]$$

and

$$V(r) = E_{0u}[(M(0) - E(u))^2]$$

where E_{0u} denotes the conditional expectation given that there are points of the process at the locations 0 and u separated by a distance r, and where $M(0)$ denotes the mark attached to the point 0.

These functions may serve as diagnostics for dependence between the points and the marks. If the points and marks are independent, then $E(r)$ and $V(r)$ should be constant (not depending on r). See Schlather et al (2004).

The argument `X` must be a point pattern (object of class `"ppp"`) or any data that are acceptable to `as.ppp`. It must be a marked point pattern with numeric marks.

The argument `r` is the vector of values for the distance r at which $k_f(r)$ is estimated.

This algorithm assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape.

Biases due to edge effects are treated in the same manner as in `Kest`. The edge corrections implemented here are

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is implemented only for rectangular and polygonal windows (not for binary masks).

translate Translation correction (Ohser, 1983). Implemented for all window geometries, but slow for complex windows.

Note that the estimator assumes the process is stationary (spatially homogeneous).

The numerator and denominator of the mark correlation function (in the expression above) are estimated using density estimation techniques. The user can choose between

`"density"` which uses the standard kernel density estimation routine `density`, and works only for evenly-spaced `r` values;

`"loess"` which uses the function `loess` in the package **modreg**;

`"sm"` which uses the function `sm.density` in the package **sm** and is extremely slow;

`"smrep"` which uses the function `sm.density` in the package **sm** and is relatively fast, but may require manual control of the smoothing parameter `hmult`.

Value

If `marks(X)` is a numeric vector, the result is an object of class `"fv"` (see `fv.object`). If `marks(X)` is a data frame, the result is a list of objects of class `"fv"`, one for each column of marks.

An object of class `"fv"` is essentially a data frame containing numeric columns

`r`	the values of the argument r at which the function $E(r)$ or $V(r)$ has been estimated
`theo`	the theoretical, constant value of $E(r)$ or $V(r)$ when the marks attached to different points are independent

together with a column or columns named `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $E(r)$ or $V(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Schlather, M. and Ribeiro, P. and Diggle, P. (2004) Detecting dependence between marks and locations of marked point processes. *Journal of the Royal Statistical Society, series B* **66** (2004) 79-83.

See Also

Mark correlation `markcorr`, mark variogram `markvario` for numeric marks.

Mark connection function `markconnect` and multitype K-functions `Kcross`, `Kdot` for factor-valued marks.

Examples

```
plot(Emark(spruces))
E <- Emark(spruces, method="density", kernel="epanechnikov")
plot(Vmark(spruces))
```

`emend` *Force Model to be Valid*

Description

Check whether a model is valid, and if not, find the nearest model which is valid.

Usage

```
emend(object, ...)
```

Arguments

`object`	A statistical model, belonging to some class.
`...`	Arguments passed to methods.

Details

The function `emend` is generic, and has methods for several classes of statistical models in the **spatstat** package (mostly point process models). Its purpose is to check whether a given model is valid (for example, that none of the model parameters are `NA`) and, if not, to find the nearest model which is valid.

See the methods for more information.

Value

Another model of the same kind.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

emend.ppm, emend.lppm, valid.

---

emend.ppm *Force Point Process Model to be Valid*

Description

Ensures that a fitted point process model satisfies the integrability conditions for existence of the point process.

Usage

```
project.ppm(object, ..., fatal=FALSE, trace=FALSE)

## S3 method for class 'ppm'
emend(object, ..., fatal=FALSE, trace=FALSE)
```

Arguments

`object`	Fitted point process model (object of class "ppm").
`...`	Ignored.
`fatal`	Logical value indicating whether to generate an error if the model cannot be projected to a valid model.
`trace`	Logical value indicating whether to print a trace of the decision process.

Details

The functions `emend.ppm` and `project.ppm` are identical: `emend.ppm` is a method for the generic `emend`, while `project.ppm` is an older name for the same function.

The purpose of the function is to ensure that a fitted model is valid.

The model-fitting function `ppm` fits Gibbs point process models to point pattern data. By default, the fitted model returned by `ppm` may not actually exist as a point process.

First, some of the fitted coefficients of the model may be `NA` or infinite values. This usually occurs when the data are insufficient to estimate all the parameters. The model is said to be *unidentifiable* or *confounded*.

Second, unlike a regression model, which is well-defined for any finite values of the fitted regression coefficients, a Gibbs point process model is only well-defined if the fitted interaction parameters satisfy some constraints. A famous example is the Strauss process (see `Strauss`) which exists only when the interaction parameter γ is less than or equal to 1. For values $\gamma > 1$, the probability density is not integrable and the process does not exist (and cannot be simulated).

By default, `ppm` does not enforce the constraint that a fitted Strauss process (for example) must satisfy $\gamma \leq 1$. This is because a fitted parameter value of $\gamma > 1$ could be useful information for data analysis, as it indicates that the Strauss model is not appropriate, and suggests a clustered model should be fitted.

The function `emend.ppm` or `project.ppm` modifies the model `object` so that the model is valid. It identifies the terms in the model `object` that are associated with illegal parameter values (i.e. parameter values which are either `NA`, infinite, or outside their permitted range). It considers all possible sub-models of object obtained by deleting one or more of these terms. It identifies which of these submodels are valid, and chooses the valid submodel with the largest pseudolikelihood. The result of emend.ppm or project.ppm is the true maximum pseudolikelihood fit to the data.

For large datasets or complex models, the algorithm used in emend.ppm or project.ppm may be time-consuming, because it takes time to compute all the sub-models. A faster, approximate algorithm can be applied by setting spatstat.options(project.fast=TRUE). This produces a valid submodel, which may not be the maximum pseudolikelihood submodel.

Use the function valid.ppm to check whether a fitted model object specifies a well-defined point process.

Use the expression all(is.finite(coef(object))) to determine whether all parameters are identifiable.

Value

Another point process model (object of class "ppm").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, valid.ppm, emend, spatstat.options

Examples

```
fit <- ppm(redwood, ~1, Strauss(0.1))
coef(fit)
fit2 <- emend(fit)
coef(fit2)
```

---

endpoints.psp *Endpoints of Line Segment Pattern*

Description

Extracts the endpoints of each line segment in a line segment pattern.

Usage

```
endpoints.psp(x, which="both")
```

Arguments

x	A line segment pattern (object of class "psp").
which	String specifying which endpoint or endpoints should be returned. See Details.

Details

This function extracts one endpoint, or both endpoints, from each of the line segments in `x`, and returns these points as a point pattern object.

The argument `which` determines which endpoint or endpoints of each line segment should be returned:

`which="both"` (the default): both endpoints of each line segment are returned. The result is a point pattern with twice as many points as there are line segments in `x`.

`which="first"` select the first endpoint of each line segment (returns the points with coordinates `x$ends$x0, x$ends$y0`).

`which="second"` select the second endpoint of each line segment (returns the points with coordinates `x$ends$x1, x$ends$y1`).

`which="left"` select the left-most endpoint (the endpoint with the smaller x coordinate) of each line segment.

`which="right"` select the right-most endpoint (the endpoint with the greater x coordinate) of each line segment.

`which="lower"` select the lower endpoint (the endpoint with the smaller y coordinate) of each line segment.

`which="upper"` select the upper endpoint (the endpoint with the greater y coordinate) of each line segment.

The result is a point pattern. It also has an attribute `"id"` which is an integer vector identifying the segment which contributed each point.

Value

Point pattern (object of class `"ppp"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`psp.object`, `ppp.object`, `midpoints.psp`

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(a)
b <- endpoints.psp(a, "left")
plot(b, add=TRUE)
```

envelope *Simulation Envelopes of Summary Function*

Description

Computes simulation envelopes of a summary function.

Usage

```
  envelope(Y, fun, ...)

  ## S3 method for class 'ppp'
envelope(Y, fun=Kest, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs,
  simulate=NULL,  fix.n=FALSE, fix.marks=FALSE,
  verbose=TRUE, clipdata=TRUE,
  transform=NULL, global=FALSE, ginterval=NULL, use.theory=NULL,
  alternative=c("two.sided", "less", "greater"),
  scale=NULL, clamp=FALSE,
  savefuns=FALSE, savepatterns=FALSE,
  nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL, maxnerr=nsim,
  do.pwrong=FALSE, envir.simul=NULL)

  ## S3 method for class 'ppm'
envelope(Y, fun=Kest, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs,
  simulate=NULL, fix.n=FALSE, fix.marks=FALSE,
  verbose=TRUE, clipdata=TRUE,
  start=NULL, control=update(default.rmhcontrol(Y), nrep=nrep), nrep=1e5,
  transform=NULL, global=FALSE, ginterval=NULL, use.theory=NULL,
  alternative=c("two.sided", "less", "greater"),
  scale=NULL, clamp=FALSE,
  savefuns=FALSE, savepatterns=FALSE,
  nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL, maxnerr=nsim,
  do.pwrong=FALSE, envir.simul=NULL)

  ## S3 method for class 'kppm'
envelope(Y, fun=Kest, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs,
  simulate=NULL,
  verbose=TRUE, clipdata=TRUE,
  transform=NULL, global=FALSE, ginterval=NULL, use.theory=NULL,
  alternative=c("two.sided", "less", "greater"),
  scale=NULL, clamp=FALSE,
  savefuns=FALSE, savepatterns=FALSE,
  nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL, maxnerr=nsim,
  do.pwrong=FALSE, envir.simul=NULL)
```

Arguments

`Y`	Object containing point pattern data. A point pattern (object of class "ppp") or a fitted point process model (object of class "ppm" or "kppm").

`fun`	Function that computes the desired summary statistic for a point pattern.
`nsim`	Number of simulated point patterns to be generated when computing the envelopes.
`nrank`	Integer. Rank of the envelope value amongst the `nsim` simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.
`...`	Extra arguments passed to `fun`.
`funargs`	A list, containing extra arguments to be passed to `fun`.
`funYargs`	Optional. A list, containing extra arguments to be passed to `fun` when applied to the original data `Y` only.
`simulate`	Optional. Specifies how to generate the simulated point patterns. If `simulate` is an expression in the R language, then this expression will be evaluated `nsim` times, to obtain `nsim` point patterns which are taken as the simulated patterns from which the envelopes are computed. If `simulate` is a list of point patterns, then the entries in this list will be treated as the simulated patterns from which the envelopes are computed. Alternatively `simulate` may be an object produced by the `envelope` command: see Details.
`fix.n`	Logical. If `TRUE`, simulated patterns will have the same number of points as the original data pattern. This option is currently not available for `envelope.kppm`.
`fix.marks`	Logical. If `TRUE`, simulated patterns will have the same number of points *and* the same marks as the original data pattern. In a multitype point pattern this means that the simulated patterns will have the same number of points *of each type* as the original data. This option is currently not available for `envelope.kppm`.
`verbose`	Logical flag indicating whether to print progress reports during the simulations.
`clipdata`	Logical flag indicating whether the data point pattern should be clipped to the same window as the simulated patterns, before the summary function for the data is computed. This should usually be `TRUE` to ensure that the data and simulations are properly comparable.
`start,control`	Optional. These specify the arguments `start` and `control` of `rmh`, giving complete control over the simulation algorithm. Applicable only when `Y` is a fitted model of class `"ppm"`.
`nrep`	Number of iterations in the Metropolis-Hastings simulation algorithm. Applicable only when `Y` is a fitted model of class `"ppm"`.
`transform`	Optional. A transformation to be applied to the function values, before the envelopes are computed. An expression object (see Details).
`global`	Logical flag indicating whether envelopes should be pointwise (`global=FALSE`) or simultaneous (`global=TRUE`).
`ginterval`	Optional. A vector of length 2 specifying the interval of r values for the simultaneous critical envelopes. Only relevant if `global=TRUE`.
`use.theory`	Logical value indicating whether to use the theoretical value, computed by `fun`, as the reference value for simultaneous envelopes. Applicable only when `global=TRUE`. Default is `use.theory=TRUE` if `Y` is a point pattern, or a point process model equivalent to Complete Spatial Randomness, and `use.theory=FALSE` otherwise.
`alternative`	Character string determining whether the envelope corresponds to a two-sided test (`side="two.sided"`, the default) or a one-sided test with a lower critical boundary (`side="less"`) or a one-sided test with an upper critical boundary (`side="greater"`).

scale	Optional. Scaling function for global envelopes. A function in the R language which determines the relative scale of deviations, as a function of distance r, when computing the global envelopes. Applicable only when `global=TRUE`. Summary function values for distance `r` will be *divided* by `scale(r)` before the maximum deviation is computed. The resulting global envelopes will have width proportional to `scale(r)`.
clamp	Logical value indicating how to compute envelopes when `alternative="less"` or `alternative="greater"`. Deviations of the observed summary function from the theoretical summary function are initially evaluated as signed real numbers, with large positive values indicating consistency with the alternative hypothesis. If `clamp=FALSE` (the default), these values are not changed. If `clamp=TRUE`, any negative values are replaced by zero.
savefuns	Logical flag indicating whether to save all the simulated function values.
savepatterns	Logical flag indicating whether to save all the simulated point patterns.
nsim2	Number of extra simulated point patterns to be generated if it is necessary to use simulation to estimate the theoretical mean of the summary function. Only relevant when `global=TRUE` and the simulations are not based on CSR.
VARIANCE	Logical. If `TRUE`, critical envelopes will be calculated as sample mean plus or minus `nSD` times sample standard deviation.
nSD	Number of estimated standard deviations used to determine the critical envelopes, if `VARIANCE=TRUE`.
Yname	Character string that should be used as the name of the data point pattern `Y` when printing or plotting the results.
maxnerr	Maximum number of rejected patterns. If `fun` yields an error when applied to a simulated point pattern (for example, because the pattern is empty and `fun` requires at least one point), the pattern will be rejected and a new random point pattern will be generated. If this happens more than `maxnerr` times, the algorithm will give up.
do.pwrong	Logical. If `TRUE`, the algorithm will also estimate the true significance level of the “wrong” test (the test that declares the summary function for the data to be significant if it lies outside the *pointwise* critical boundary at any point). This estimate is printed when the result is printed.
envir.simul	Environment in which to evaluate the expression `simulate`, if not the current environment.

Details

The `envelope` command performs simulations and computes envelopes of a summary statistic based on the simulations. The result is an object that can be plotted to display the envelopes. The envelopes can be used to assess the goodness-of-fit of a point process model to point pattern data.

For the most basic use, if you have a point pattern `X` and you want to test Complete Spatial Randomness (CSR), type `plot(envelope(X, Kest,nsim=39))` to see the K function for `X` plotted together with the envelopes of the K function for 39 simulations of CSR.

The `envelope` function is generic, with methods for the classes `"ppp"`, `"ppm"` and `"kppm"` described here. There are also methods for the classes `"pp3"`, `"lpp"` and `"lppm"` which are described separately under `envelope.pp3` and `envelope.lpp`. Envelopes can also be computed from other envelopes, using `envelope.envelope`.

To create simulation envelopes, the command `envelope(Y, ...)` first generates `nsim` random point patterns in one of the following ways.

- If Y is a point pattern (an object of class "ppp") and simulate=NULL, then we generate nsim simulations of Complete Spatial Randomness (i.e. nsim simulated point patterns each being a realisation of the uniform Poisson point process) with the same intensity as the pattern Y. (If Y is a multitype point pattern, then the simulated patterns are also given independent random marks; the probability distribution of the random marks is determined by the relative frequencies of marks in Y.)
- If Y is a fitted point process model (an object of class "ppm" or "kppm") and simulate=NULL, then this routine generates nsim simulated realisations of that model.
- If simulate is supplied, then it determines how the simulated point patterns are generated. It may be either
  - an expression in the R language, typically containing a call to a random generator. This expression will be evaluated nsim times to yield nsim point patterns. For example if simulate=expression(runifpoint(100)) then each simulated pattern consists of exactly 100 independent uniform random points.
  - a list of point patterns. The entries in this list will be taken as the simulated patterns.
  - an object of class "envelope". This should have been produced by calling envelope with the argument savepatterns=TRUE. The simulated point patterns that were saved in this object will be extracted and used as the simulated patterns for the new envelope computation. This makes it possible to plot envelopes for two different summary functions based on exactly the same set of simulated point patterns.

The summary statistic fun is applied to each of these simulated patterns. Typically fun is one of the functions Kest, Gest, Fest, Jest, pcf, Kcross, Kdot, Gcross, Gdot, Jcross, Jdot, Kmulti, Gmulti, Jmulti or Kinhom. It may also be a character string containing the name of one of these functions.

The statistic fun can also be a user-supplied function; if so, then it must have arguments X and r like those in the functions listed above, and it must return an object of class "fv".

Upper and lower critical envelopes are computed in one of the following ways:

pointwise: by default, envelopes are calculated pointwise (i.e. for each value of the distance argument r), by sorting the nsim simulated values, and taking the m-th lowest and m-th highest values, where m = nrank. For example if nrank=1, the upper and lower envelopes are the pointwise maximum and minimum of the simulated values.

The pointwise envelopes are **not** "confidence bands" for the true value of the function! Rather, they specify the critical points for a Monte Carlo test (Ripley, 1981). The test is constructed by choosing a *fixed* value of r, and rejecting the null hypothesis if the observed function value lies outside the envelope *at this value of* r. This test has exact significance level alpha = 2 * nrank/(1 + nsim).

simultaneous: if global=TRUE, then the envelopes are determined as follows. First we calculate the theoretical mean value of the summary statistic (if we are testing CSR, the theoretical value is supplied by fun; otherwise we perform a separate set of nsim2 simulations, compute the average of all these simulated values, and take this average as an estimate of the theoretical mean value). Then, for each simulation, we compare the simulated curve to the theoretical curve, and compute the maximum absolute difference between them (over the interval of r values specified by ginterval). This gives a deviation value d_i for each of the nsim simulations. Finally we take the m-th largest of the deviation values, where m=nrank, and call this dcrit. Then the simultaneous envelopes are of the form lo = expected - dcrit and hi = expected + dcrit where expected is either the theoretical mean value theo (if we are testing CSR) or the estimated theoretical value mmean (if we are testing another model). The simultaneous critical envelopes have constant width 2 * dcrit.

The simultaneous critical envelopes allow us to perform a different Monte Carlo test (Ripley, 1981). The test rejects the null hypothesis if the graph of the observed function lies outside the envelope **at any value of** r. This test has exact significance level `alpha = nrank/(1 + nsim)`. This test can also be performed using `mad.test`.

based on sample moments: if `VARIANCE=TRUE`, the algorithm calculates the (pointwise) sample mean and sample variance of the simulated functions. Then the envelopes are computed as mean plus or minus `nSD` standard deviations. These envelopes do not have an exact significance interpretation. They are a naive approximation to the critical points of the Neyman-Pearson test assuming the summary statistic is approximately Normally distributed.

The return value is an object of class `"fv"` containing the summary function for the data point pattern, the upper and lower simulation envelopes, and the theoretical expected value (exact or estimated) of the summary function for the model being tested. It can be plotted using `plot.envelope`.

If `VARIANCE=TRUE` then the return value also includes the sample mean, sample variance and other quantities.

Arguments can be passed to the function `fun` through `...`. This means that you simply specify these arguments in the call to `envelope`, and they will be passed to `fun`. In particular, the argument `correction` determines the edge correction to be used to calculate the summary statistic. See the section on Edge Corrections, and the Examples.

Arguments can also be passed to the function `fun` through the list `funargs`. This mechanism is typically used if an argument of `fun` has the same name as an argument of `envelope`. The list `funargs` should contain entries of the form `name=value`, where each `name` is the name of an argument of `fun`.

There is also an option, rarely used, in which different function arguments are used when computing the summary function for the data `Y` and for the simulated patterns. If `funYargs` is given, it will be used when the summary function for the data `Y` is computed, while `funargs` will be used when computing the summary function for the simulated patterns. This option is only needed in rare cases: usually the basic principle requires that the data and simulated patterns must be treated equally, so that `funargs` and `funYargs` should be identical.

If `Y` is a fitted cluster point process model (object of class `"kppm"`), and `simulate=NULL`, then the model is simulated directly using `simulate.kppm`.

If `Y` is a fitted Gibbs point process model (object of class `"ppm"`), and `simulate=NULL`, then the model is simulated by running the Metropolis-Hastings algorithm `rmh`. Complete control over this algorithm is provided by the arguments `start` and `control` which are passed to `rmh`.

For simultaneous critical envelopes (`global=TRUE`) the following options are also useful:

`ginterval` determines the interval of r values over which the deviation between curves is calculated. It should be a numeric vector of length 2. There is a sensible default (namely, the recommended plotting interval for `fun(X)`, or the range of `r` values if `r` is explicitly specified).

`transform` specifies a transformation of the summary function `fun` that will be carried out before the deviations are computed. Such transforms are useful if `global=TRUE` or `VARIANCE=TRUE`. The `transform` must be an expression object using the symbol `.` to represent the function value (and possibly other symbols recognised by `with.fv`). For example, the conventional way to normalise the K function (Ripley, 1981) is to transform it to the L function $L(r) = \sqrt{K(r)/\pi}$ and this is implemented by setting `transform=expression(sqrt(./pi))`.

It is also possible to extract the summary functions for each of the individual simulated point patterns, by setting `savefuns=TRUE`. Then the return value also has an attribute `"simfuns"` containing all the summary functions for the individual simulated patterns. It is an `"fv"` object containing functions named `sim1, sim2, ...` representing the `nsim` summary functions.

It is also possible to save the simulated point patterns themselves, by setting savepatterns=TRUE. Then the return value also has an attribute "simpatterns" which is a list of length nsim containing all the simulated point patterns.

See plot.envelope and plot.fv for information about how to plot the envelopes.

Different envelopes can be recomputed from the same data using envelope.envelope. Envelopes can be combined using pool.envelope.

Value

An object of class "envelope" and "fv", see fv.object, which can be printed and plotted directly.

Essentially a data frame containing columns

r	the vector of values of the argument r at which the summary function fun has been estimated
obs	values of the summary function for the data point pattern
lo	lower envelope of simulations
hi	upper envelope of simulations

and *either*

theo	theoretical value of the summary function under CSR (Complete Spatial Randomness, a uniform Poisson point process) if the simulations were generated according to CSR
mmean	estimated theoretical value of the summary function, computed by averaging simulated values, if the simulations were not generated according to CSR.

Additionally, if savepatterns=TRUE, the return value has an attribute "simpatterns" which is a list containing the nsim simulated patterns. If savefuns=TRUE, the return value has an attribute "simfuns" which is an object of class "fv" containing the summary functions computed for each of the nsim simulated patterns.

Errors and warnings

An error may be generated if one of the simulations produces a point pattern that is empty, or is otherwise unacceptable to the function fun.

The upper envelope may be NA (plotted as plus or minus infinity) if some of the function values computed for the simulated point patterns are NA. Whether this occurs will depend on the function fun, but it usually happens when the simulated point pattern does not contain enough points to compute a meaningful value.

Confidence intervals

Simulation envelopes do **not** compute confidence intervals; they generate significance bands. If you really need a confidence interval for the true summary function of the point process, use lohboot. See also varblock.

Edge corrections

It is common to apply a correction for edge effects when calculating a summary function such as the K function. Typically the user has a choice between several possible edge corrections. In a call to envelope, the user can specify the edge correction to be applied in fun, using the argument correction. See the Examples below.

Summary functions in spatstat Summary functions that are available in **spatstat**, such as `Kest`, `Gest` and `pcf`, have a standard argument called `correction` which specifies the name of one or more edge corrections.

The list of available edge corrections is different for each summary function, and may also depend on the kind of window in which the point pattern is recorded. In the case of `Kest` (the default and most frequently used value of `fun`) the best edge correction is Ripley's isotropic correction if the window is rectangular or polygonal, and the translation correction if the window is a binary mask. See the help files for the individual functions for more information.

All the summary functions in **spatstat** recognise the option `correction="best"` which gives the "best" (most accurate) available edge correction for that function.

In a call to `envelope`, if `fun` is one of the summary functions provided in **spatstat**, then the default is `correction="best"`. This means that *by default, the envelope will be computed using the "best" available edge correction*.

The user can override this default by specifying the argument `correction`. For example the computation can be accelerated by choosing another edge correction which is less accurate than the "best" one, but faster to compute.

User-written summary functions If `fun` is a function written by the user, then `envelope` has to guess what to do.

If `fun` has an argument called `correction`, or has `...` arguments, then `envelope` assumes that the function can handle a correction argument. To compute the envelope, `fun` will be called with a `correction` argument. The default is `correction="best"`, unless overridden in the call to `envelope`.

Otherwise, if `fun` does not have an argument called `correction` and does not have `...` arguments, then `envelope` assumes that the function *cannot* handle a correction argument. To compute the envelope, `fun` is called without a correction argument.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Diggle, P.J., Hardegen, A., Lawrence, T., Milne, R.K. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84** (3) 477–489.

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Arnold, 2003.

Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

`dclf.test`, `mad.test` for envelope-based tests.

`fv.object`, `plot.envelope`, `plot.fv`, `envelope.envelope`, `pool.envelope` for handling envelopes. There are also methods for `print` and `summary`.

`Kest`, `Gest`, `Fest`, `Jest`, `pcf`, `ppp`, `ppm`, `default.expand`

Examples

```
 X <- simdat

 # Envelope of K function under CSR
 ## Not run: 
 plot(envelope(X))

## End(Not run)


 # Translation edge correction (this is also FASTER):
 ## Not run: 
 plot(envelope(X, correction="translate"))

## End(Not run)


# Global envelopes
 ## Not run: 
 plot(envelope(X, Lest, global=TRUE))
 plot(envelope(X, Kest,  global=TRUE, scale=function(r) { r }))

## End(Not run)


 # Envelope of K function for simulations from Gibbs model
 ## Not run: 
 fit <- ppm(cells ~1, Strauss(0.05))
 plot(envelope(fit))
 plot(envelope(fit), global=TRUE)

## End(Not run)


 # Envelope of K function for simulations from cluster model
 fit <- kppm(redwood ~1, "Thomas")
 ## Not run: 
 plot(envelope(fit, Gest))
 plot(envelope(fit, Gest, global=TRUE))

## End(Not run)


 # Envelope of G function under CSR
 ## Not run: 
 plot(envelope(X, Gest))

## End(Not run)


 # Envelope of L function under CSR
 #  L(r) = sqrt(K(r)/pi)
 ## Not run: 
  E <- envelope(X, Kest)
  plot(E, sqrt(./pi) ~ r)
```

```
## End(Not run)


 # Simultaneous critical envelope for L function
 # (alternatively, use Lest)
 ## Not run:
  plot(envelope(X, Kest, transform=expression(sqrt(./pi)), global=TRUE))

## End(Not run)


 ## One-sided envelope
 ## Not run:
  plot(envelope(X, Lest, alternative="less"))

## End(Not run)


 # How to pass arguments needed to compute the summary functions:
 # We want envelopes for Jcross(X, "A", "B")
 # where "A" and "B" are types of points in the dataset 'demopat'

 data(demopat)
 ## Not run:
 plot(envelope(demopat, Jcross, i="A", j="B"))

## End(Not run)


 # Use of `simulate'
 ## Not run:
 plot(envelope(cells, Gest, simulate=expression(runifpoint(42))))
 plot(envelope(cells, Gest, simulate=expression(rMaternI(100,0.02))))

## End(Not run)


 # Envelope under random toroidal shifts
 data(amacrine)
 ## Not run:
 plot(envelope(amacrine, Kcross, i="on", j="off",
               simulate=expression(rshift(amacrine, radius=0.25))))

## End(Not run)

 # Envelope under random shifts with erosion
 ## Not run:
 plot(envelope(amacrine, Kcross, i="on", j="off",
              simulate=expression(rshift(amacrine, radius=0.1, edge="erode"))))

## End(Not run)

 # Envelope of INHOMOGENEOUS K-function with fitted trend

 # The following is valid.
 # Setting lambda=fit means that the fitted model is re-fitted to
 # each simulated pattern to obtain the intensity estimates for Kinhom.
```

```
 # (lambda=NULL would also be valid)

 fit <- kppm(redwood ~1, clusters="MatClust")
 ## Not run:
    plot(envelope(fit, Kinhom, lambda=fit, nsim=19))

## End(Not run)


 # Note that the principle of symmetry, essential to the validity of
 # simulation envelopes, requires that both the observed and
 # simulated patterns be subjected to the same method of intensity
 # estimation. In the following example it would be incorrect to set the
 # argument 'lambda=red.dens' in the envelope command, because this
 # would mean that the inhomogeneous K functions of the simulated
 # patterns would be computed using the intensity function estimated
 # from the original redwood data, violating the symmetry.  There is
 # still a concern about the fact that the simulations are generated
 # from a model that was fitted to the data; this is only a problem in
 # small datasets.

## Not run:
 red.dens <- density(redwood, sigma=bw.diggle)
 plot(envelope(redwood, Kinhom, sigma=bw.diggle,
         simulate=expression(rpoispp(red.dens))))

## End(Not run)

 # Precomputed list of point patterns
## Not run:
 nX <- npoints(X)
 PatList <- list()
 for(i in 1:19) PatList[[i]] <- runifpoint(nX)
 E <- envelope(X, Kest, nsim=19, simulate=PatList)

## End(Not run)


# re-using the same point patterns
## Not run:
 EK <- envelope(X, Kest, savepatterns=TRUE)
 EG <- envelope(X, Gest, simulate=EK)

## End(Not run)
```

---

envelope.envelope *Recompute Envelopes*

Description

Given a simulation envelope (object of class "envelope"), compute another envelope from the same simulation data using different parameters.

Usage

```
## S3 method for class 'envelope'
envelope(Y, fun = NULL, ...,
                              transform=NULL, global=FALSE, VARIANCE=FALSE)
```

Arguments

Y
: A simulation envelope (object of class "envelope").

fun
: Optional. Summary function to be applied to the simulated point patterns.

...,transform,global,VARIANCE
: Parameters controlling the type of envelope that is re-computed. See envelope.

Details

This function can be used to re-compute a simulation envelope from previously simulated data, using different parameter settings for the envelope: for example, a different significance level, or a global envelope instead of a pointwise envelope.

The function envelope is generic. This is the method for the class "envelope".

The argument Y should be a simulation envelope (object of class "envelope") produced by any of the methods for envelope. Additionally, Y must contain either

- the simulated point patterns that were used to create the original envelope (so Y should have been created by calling envelope with savepatterns=TRUE);
- the summary functions of the simulated point patterns that were used to create the original envelope (so Y should have been created by calling envelope with savefuns=TRUE).

If the argument fun is given, it should be a summary function that can be applied to the simulated point patterns that were used to create Y. The envelope of the summary function fun for these point patterns will be computed using the parameters specified in ....

If fun is not given, then:

- If Y contains the summary functions that were used to compute the original envelope, then the new envelope will be computed from these original summary functions.
- Otherwise, if Y contains the simulated point patterns. then the K function Kest will be applied to each of these simulated point patterns, and the new envelope will be based on the K functions.

The new envelope will be computed using the parameters specified in ....

See envelope for a full list of envelope parameters. Frequently-used parameters include nrank and nsim (to change the number of simulations used and the significance level of the envelope), global (to change from pointwise to global envelopes) and VARIANCE (to compute the envelopes from the sample moments instead of the ranks).

Value

An envelope (object of class "envelope".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

envelope

Examples

```
E <- envelope(cells, Kest, nsim=19, savefuns=TRUE, savepatterns=TRUE)
E2 <- envelope(E, nrank=2)
Eg <- envelope(E, global=TRUE)
EG <- envelope(E, Gest)
EL <- envelope(E, transform=expression(sqrt(./pi)))
```

---

envelope.lpp *Envelope for Point Patterns on Linear Network*

Description

Enables envelopes to be computed for point patterns on a linear network.

Usage

```
  ## S3 method for class 'lpp'
envelope(Y, fun=linearK, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs,
  simulate=NULL, fix.n=FALSE, fix.marks=FALSE, verbose=TRUE,
  transform=NULL,global=FALSE,ginterval=NULL,use.theory=NULL,
  alternative=c("two.sided", "less", "greater"),
  scale=NULL, clamp=FALSE,
  savefuns=FALSE, savepatterns=FALSE,
  nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL,
  do.pwrong=FALSE, envir.simul=NULL)


  ## S3 method for class 'lppm'
envelope(Y, fun=linearK, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs,
  simulate=NULL, fix.n=FALSE, fix.marks=FALSE, verbose=TRUE,
  transform=NULL,global=FALSE,ginterval=NULL,use.theory=NULL,
  alternative=c("two.sided", "less", "greater"),
  scale=NULL, clamp=FALSE,
  savefuns=FALSE, savepatterns=FALSE,
  nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL,
  do.pwrong=FALSE, envir.simul=NULL)
```

Arguments

Y	A point pattern on a linear network (object of class "lpp") or a fitted point process model on a linear network (object of class "lppm").
fun	Function that is to be computed for each simulated pattern.
nsim	Number of simulations to perform.
nrank	Integer. Rank of the envelope value amongst the nsim simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.

...	Extra arguments passed to fun.
funargs	A list, containing extra arguments to be passed to fun.
funYargs	Optional. A list, containing extra arguments to be passed to fun when applied to the original data Y only.
simulate	Optional. Specifies how to generate the simulated point patterns. If simulate is an expression in the R language, then this expression will be evaluated nsim times, to obtain nsim point patterns which are taken as the simulated patterns from which the envelopes are computed. If simulate is a list of point patterns, then the entries in this list will be treated as the simulated patterns from which the envelopes are computed. Alternatively simulate may be an object produced by the envelope command: see Details.
fix.n	Logical. If TRUE, simulated patterns will have the same number of points as the original data pattern.
fix.marks	Logical. If TRUE, simulated patterns will have the same number of points *and* the same marks as the original data pattern. In a multitype point pattern this means that the simulated patterns will have the same number of points *of each type* as the original data.
verbose	Logical flag indicating whether to print progress reports during the simulations.
transform	Optional. A transformation to be applied to the function values, before the envelopes are computed. An expression object (see Details).
global	Logical flag indicating whether envelopes should be pointwise (global=FALSE) or simultaneous (global=TRUE).
ginterval	Optional. A vector of length 2 specifying the interval of r values for the simultaneous critical envelopes. Only relevant if global=TRUE.
use.theory	Logical value indicating whether to use the theoretical value, computed by fun, as the reference value for simultaneous envelopes. Applicable only when global=TRUE.
alternative	Character string determining whether the envelope corresponds to a two-sided test (side="two.sided", the default) or a one-sided test with a lower critical boundary (side="less") or a one-sided test with an upper critical boundary (side="greater").
scale	Optional. Scaling function for global envelopes. A function in the R language which determines the relative scale of deviations, as a function of distance r, when computing the global envelopes. Applicable only when global=TRUE. Summary function values for distance r will be *divided* by scale(r) before the maximum deviation is computed. The resulting global envelopes will have width proportional to scale(r).
clamp	Logical value indicating how to compute envelopes when alternative="less" or alternative="greater". Deviations of the observed summary function from the theoretical summary function are initially evaluated as signed real numbers, with large positive values indicating consistency with the alternative hypothesis. If clamp=FALSE (the default), these values are not changed. If clamp=TRUE, any negative values are replaced by zero.
savefuns	Logical flag indicating whether to save all the simulated function values.
savepatterns	Logical flag indicating whether to save all the simulated point patterns.
nsim2	Number of extra simulated point patterns to be generated if it is necessary to use simulation to estimate the theoretical mean of the summary function. Only relevant when global=TRUE and the simulations are not based on CSR.

VARIANCE	Logical. If TRUE, critical envelopes will be calculated as sample mean plus or minus nSD times sample standard deviation.
nSD	Number of estimated standard deviations used to determine the critical envelopes, if VARIANCE=TRUE.
Yname	Character string that should be used as the name of the data point pattern Y when printing or plotting the results.
do.pwrong	Logical. If TRUE, the algorithm will also estimate the true significance level of the "wrong" test (the test that declares the summary function for the data to be significant if it lies outside the *pointwise* critical boundary at any point). This estimate is printed when the result is printed.
envir.simul	Environment in which to evaluate the expression simulate, if not the current environment.

Details

This is a method for the generic function envelope applicable to point patterns on a linear network.

The argument Y can be either a point pattern on a linear network, or a fitted point process model on a linear network. The function fun will be evaluated for the data and also for nsim simulated point patterns on the same linear network. The upper and lower envelopes of these evaluated functions will be computed as described in envelope.

The type of simulation is determined as follows.

- if Y is a point pattern (object of class "lpp") and simulate is missing or NULL, then random point patterns will be generated according to a Poisson point process on the linear network on which Y is defined, with intensity estimated from Y.
- if Y is a fitted point process model (object of class "lppm") and simulate is missing or NULL, then random point patterns will be generated by simulating from the fitted model.
- If simulate is present, it should be an expression that can be evaluated to yield random point patterns on the same linear network as Y.

The function fun should accept as its first argument a point pattern on a linear network (object of class "lpp") and should have another argument called r or a ... argument.

Value

Function value table (object of class "fv") with additional information, as described in envelope.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Ang, Q.W. (2010) *Statistical methodology for events on a network.* Master's thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

Okabe, A. and Yamada, I. (2001) The K-function method on a network and its computational implementation. *Geographical Analysis* **33**, 271-290.

See Also

envelope, linearK

Examples

```
if(interactive()) {
  ns <- 39
  np <- 40
} else { ns <- np <- 3 }
X <- runiflpp(np, simplenet)

# uniform Poisson: random numbers of points
envelope(X, nsim=ns)

# uniform Poisson: conditional on observed number of points
envelope(X, fix.n=TRUE, nsim=ns)

# nonuniform Poisson
fit <- lppm(X ~x)
envelope(fit, nsim=ns)

#multitype
marks(X) <- sample(letters[1:2], np, replace=TRUE)
envelope(X, nsim=ns)
```

---

envelope.pp3 *Simulation Envelopes of Summary Function for 3D Point Pattern*

Description

Computes simulation envelopes of a summary function for a three-dimensional point pattern.

Usage

```
  ## S3 method for class 'pp3'
envelope(Y, fun=K3est, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs, simulate=NULL, verbose=TRUE,
  transform=NULL,global=FALSE,ginterval=NULL,use.theory=NULL,
  alternative=c("two.sided", "less", "greater"),
  scale=NULL, clamp=FALSE,
  savefuns=FALSE, savepatterns=FALSE,
  nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL, maxnerr=nsim,
  do.pwrong=FALSE, envir.simul=NULL)
```

Arguments

Y	A three-dimensional point pattern (object of class "pp3").
fun	Function that computes the desired summary statistic for a 3D point pattern.
nsim	Number of simulated point patterns to be generated when computing the envelopes.
nrank	Integer. Rank of the envelope value amongst the nsim simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.

`...`	Extra arguments passed to `fun`.
`funargs`	A list, containing extra arguments to be passed to `fun`.
`funYargs`	Optional. A list, containing extra arguments to be passed to `fun` when applied to the original data `Y` only.
`simulate`	Optional. Specifies how to generate the simulated point patterns. If `simulate` is an expression in the R language, then this expression will be evaluated `nsim` times, to obtain `nsim` point patterns which are taken as the simulated patterns from which the envelopes are computed. If `simulate` is a list of point patterns, then the entries in this list will be treated as the simulated patterns from which the envelopes are computed. Alternatively `simulate` may be an object produced by the `envelope` command: see Details.
`verbose`	Logical flag indicating whether to print progress reports during the simulations.
`transform`	Optional. A transformation to be applied to the function values, before the envelopes are computed. An expression object (see Details).
`global`	Logical flag indicating whether envelopes should be pointwise (`global=FALSE`) or simultaneous (`global=TRUE`).
`ginterval`	Optional. A vector of length 2 specifying the interval of r values for the simultaneous critical envelopes. Only relevant if `global=TRUE`.
`use.theory`	Logical value indicating whether to use the theoretical value, computed by `fun`, as the reference value for simultaneous envelopes. Applicable only when `global=TRUE`.
`alternative`	Character string determining whether the envelope corresponds to a two-sided test (`side="two.sided"`, the default) or a one-sided test with a lower critical boundary (`side="less"`) or a one-sided test with an upper critical boundary (`side="greater"`).
`scale`	Optional. Scaling function for global envelopes. A function in the R language which determines the relative scale of deviations, as a function of distance r, when computing the global envelopes. Applicable only when `global=TRUE`. Summary function values for distance `r` will be *divided* by `scale(r)` before the maximum deviation is computed. The resulting global envelopes will have width proportional to `scale(r)`.
`clamp`	Logical value indicating how to compute envelopes when `alternative="less"` or `alternative="greater"`. Deviations of the observed summary function from the theoretical summary function are initially evaluated as signed real numbers, with large positive values indicating consistency with the alternative hypothesis. If `clamp=FALSE` (the default), these values are not changed. If `clamp=TRUE`, any negative values are replaced by zero.
`savefuns`	Logical flag indicating whether to save all the simulated function values.
`savepatterns`	Logical flag indicating whether to save all the simulated point patterns.
`nsim2`	Number of extra simulated point patterns to be generated if it is necessary to use simulation to estimate the theoretical mean of the summary function. Only relevant when `global=TRUE` and the simulations are not based on CSR.
`VARIANCE`	Logical. If `TRUE`, critical envelopes will be calculated as sample mean plus or minus `nSD` times sample standard deviation.
`nSD`	Number of estimated standard deviations used to determine the critical envelopes, if `VARIANCE=TRUE`.
`Yname`	Character string that should be used as the name of the data point pattern `Y` when printing or plotting the results.

`maxnerr`	Maximum number of rejected patterns. If `fun` yields an error when applied to a simulated point pattern (for example, because the pattern is empty and `fun` requires at least one point), the pattern will be rejected and a new random point pattern will be generated. If this happens more than `maxnerr` times, the algorithm will give up.
`do.pwrong`	Logical. If `TRUE`, the algorithm will also estimate the true significance level of the "wrong" test (the test that declares the summary function for the data to be significant if it lies outside the *pointwise* critical boundary at any point). This estimate is printed when the result is printed.
`envir.simul`	Environment in which to evaluate the expression `simulate`, if not the current environment.

Details

The `envelope` command performs simulations and computes envelopes of a summary statistic based on the simulations. The result is an object that can be plotted to display the envelopes. The envelopes can be used to assess the goodness-of-fit of a point process model to point pattern data.

The `envelope` function is generic, with methods for the classes `"ppp"`, `"ppm"` and `"kppm"` described in the help file for `envelope`. This function `envelope.pp3` is the method for three-dimensional point patterns (objects of class `"pp3"`).

For the most basic use, if you have a 3D point pattern `X` and you want to test Complete Spatial Randomness (CSR), type `plot(envelope(X, K3est,nsim=39))` to see the three-dimensional K function for `X` plotted together with the envelopes of the three-dimensional K function for 39 simulations of CSR.

To create simulation envelopes, the command `envelope(Y, ...)` first generates `nsim` random point patterns in one of the following ways.

- If `simulate=NULL`, then we generate `nsim` simulations of Complete Spatial Randomness (i.e. `nsim` simulated point patterns each being a realisation of the uniform Poisson point process) with the same intensity as the pattern `Y`.
- If `simulate` is supplied, then it determines how the simulated point patterns are generated. See `envelope` for details.

The summary statistic `fun` is applied to each of these simulated patterns. Typically `fun` is one of the functions `K3est`, `G3est`, `F3est` or `pcf3est`. It may also be a character string containing the name of one of these functions.

For further information, see the documentation for `envelope`.

Value

A function value table (object of class `"fv"`) which can be plotted directly. See `envelope` for further details.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Baddeley, A.J, Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.

See Also

pp3, rpoispp3, K3est, G3est, F3est, pcf3est.

Examples

```
   X <- rpoispp3(20, box3())
 ## Not run:
 plot(envelope(X, nsim=39))

## End(Not run)
```

---

envelopeArray *Array of Simulation Envelopes of Summary Function*

Description

Compute an array of simulation envelopes using a summary function that returns an array of curves.

Usage

```
envelopeArray(X, fun, ..., dataname = NULL, verb = FALSE, reuse = TRUE)
```

Arguments

X	Object containing point pattern data. A point pattern (object of class "ppp", "lpp", "pp3" or "ppx") or a fitted point process model (object of class "ppm", "kppm" or "lppm").
fun	Function that computes the desired summary statistic for a point pattern. The result of fun should be a function array (object of class "fasp").
...	Arguments passed to envelope to control the simulations, or passed to fun when evaluating the function.
dataname	Optional character string name for the data.
verb	Logical value indicating whether to print progress reports.
reuse	Logical value indicating whether the envelopes in each panel should be based on the same set of simulated patterns (reuse=TRUE, the default) or on different, independent sets of simulated patterns (reuse=FALSE).

Details

This command is the counterpart of envelope when the function fun that is evaluated on each simulated point pattern will return an object of class "fasp" representing an array of summary functions.

Simulated point patterns are generated according to the rules described for envelope. In brief, if X is a point pattern, the algorithm generates simulated point patterns of the same kind, according to complete spatial randomness. If X is a fitted model, the algorithm generates simulated point patterns according to this model.

For each simulated point pattern Y, the function fun is invoked. The result Z <- fun(Y, ...) should be an object of class "fasp" representing an array of summary functions. The dimensions of the array Z should be the same for each simulated pattern Y.

This algorithm finds the simulation envelope of the summary functions in each cell of the array.

Value

An object of class "fasp" representing an array of envelopes.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

envelope, alltypes.

Examples

```
A <- envelopeArray(finpines, markcrosscorr, nsim=9)
plot(A)
```

---

eroded.areas *Areas of Morphological Erosions*

Description

Computes the areas of successive morphological erosions of a window.

Usage

```
eroded.areas(w, r, subset=NULL)
```

Arguments

w	A window.
r	Numeric vector of radii at which erosions will be performed.
subset	Optional window inside which the areas should be computed.

Details

This function computes the areas of the erosions of the window w by each of the radii r[i].

The morphological erosion of a set W by a distance $r > 0$ is the subset consisting of all points $x \in W$ such that the distance from x to the boundary of W is greater than or equal to r. In other words it is the result of trimming a margin of width r off the set W.

The argument r should be a vector of positive numbers. The argument w should be a window (an object of class "owin", see owin.object for details) or can be given in any format acceptable to as.owin().

Unless w is a rectangle, the computation is performed using a pixel raster approximation.

To compute the eroded window itself, use erosion.

Value

Numeric vector, of the same length as r, giving the areas of the successive erosions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin, as.owin, erosion

Examples

```
w <- owin(c(0,1),c(0,1))
a <- eroded.areas(w, seq(0.01,0.49,by=0.01))
```

---

erosion *Morphological Erosion by a Disc*

Description

Perform morphological erosion of a window, a line segment pattern or a point pattern by a disc.

Usage

```
 erosion(w, r, ...)
 ## S3 method for class 'owin'
erosion(w, r, shrink.frame=TRUE, ...,
   strict=FALSE, polygonal=NULL)
 ## S3 method for class 'ppp'
erosion(w, r,...)
 ## S3 method for class 'psp'
erosion(w, r,...)
```

Arguments

w	A window (object of class "owin" or a line segment pattern (object of class "psp") or a point pattern (object of class "ppp").
r	positive number: the radius of erosion.
shrink.frame	logical: if TRUE, erode the bounding rectangle as well.
...	extra arguments to as.mask controlling the pixel resolution, if pixel approximation is used.
strict	Logical flag determining the fate of boundary pixels, if pixel approximation is used. See details.
polygonal	Logical flag indicating whether to compute a polygonal approximation to the erosion (polygonal=TRUE) or a pixel grid approximation (polygonal=FALSE).

Details

The morphological erosion of a set W by a distance $r > 0$ is the subset consisting of all points $x \in W$ such that the distance from x to the boundary of W is greater than or equal to r. In other words it is the result of trimming a margin of width r off the set W.

If `polygonal=TRUE` then a polygonal approximation to the erosion is computed. If `polygonal=FALSE` then a pixel approximation to the erosion is computed from the distance map of `w`. The arguments `"..."` are passed to `as.mask` to control the pixel resolution. The erosion consists of all pixels whose distance from the boundary of `w` is strictly greater than `r` (if `strict=TRUE`) or is greater than or equal to `r` (if `strict=FALSE`).

When `w` is a window, the default (when `polygonal=NULL`) is to compute a polygonal approximation if `w` is a rectangle or polygonal window, and to compute a pixel approximation if `w` is a window of type `"mask"`.

If `shrink.frame` is false, the resulting window is given the same outer, bounding rectangle as the original window `w`. If `shrink.frame` is true, the original bounding rectangle is also eroded by the same distance `r`.

To simply compute the area of the eroded window, use `eroded.areas`.

Value

If `r > 0`, an object of class `"owin"` representing the eroded region (or `NULL` if this region is empty). If `r=0`, the result is identical to `w`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<r.turner@auckland.ac.nz>` and Ege Rubak `<rubak@math.aau.dk>`

See Also

`dilation` for the opposite operation.

`erosionAny` for morphological erosion using any shape.

`owin`, `as.owin`, `eroded.areas`

Examples

```
plot(letterR, main="erosion(letterR, 0.2)")
plot(erosion(letterR, 0.2), add=TRUE, col="red")
```

`erosionAny` *Morphological Erosion of Windows*

Description

Compute the morphological erosion of one spatial window by another.

Usage

```
erosionAny(A, B)

A %(-)% B
```

Arguments

`A,B`	Windows (objects of class "owin").

Details

The operator `A %(-)% B` and function `erosionAny(A,B)` are synonymous: they both compute the morphological erosion of the window `A` by the window `B`.

The morphological erosion $A \ominus B$ of region A by region B is the spatial region consisting of all vectors z such that, when B is shifted by the vector z, the result is a subset of A.

Equivalently

$$A \ominus B = ((A^c \oplus (-B))^c$$

where $\oplus$ is the Minkowski sum, A^c denotes the set complement, and $(-B)$ is the reflection of B through the origin, consisting of all vectors $-b$ where b is a point in B.

If `B` is a disc of radius `r`, then `erosionAny(A, B)` is equivalent to `erosion(A, r)`. See `erosion`.

The algorithm currently computes the result as a polygonal window using the **polyclip** library. It will be quite slow if applied to binary mask windows.

Value

Another window (object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`erosion`, `MinkowskiSum`

Examples

```
B <- square(c(-0.1, 0.1))
RminusB <- letterR %(-)% B
FR <- grow.rectangle(Frame(letterR), 0.3)
plot(FR, main="", type="n")
plot(letterR, add=TRUE, lwd=2, hatch=TRUE, box=FALSE)
plot(RminusB, add=TRUE, col="blue", box=FALSE)
plot(shift(B, vec=c(3.49, 2.98)),
     add=TRUE, border="red", lwd=2)
```

`eval.fasp` *Evaluate Expression Involving Function Arrays*

Description

Evaluates any expression involving one or more function arrays (`fasp` objects) and returns another function array.

Usage

```
eval.fasp(expr, envir, dotonly=TRUE)
```

Arguments

expr	An expression involving the names of objects of class "fasp".
envir	Optional. The environment in which to evaluate the expression, or a named list containing "fasp" objects to be used in the expression.
dotonly	Logical. Passed to eval.fv.

Details

This is a wrapper to make it easier to perform pointwise calculations with the arrays of summary functions used in spatial statistics.

A function array (object of class "fasp") can be regarded as a matrix whose entries are functions. Objects of this kind are returned by the command alltypes.

Suppose X is an object of class "fasp". Then eval.fasp(X+3) effectively adds 3 to the value of every function in the array X, and returns the resulting object.

Suppose X and Y are two objects of class "fasp" which are compatible (for example the arrays must have the same dimensions). Then eval.fasp(X + Y) will add the corresponding functions in each cell of the arrays X and Y, and return the resulting array of functions.

Suppose X is an object of class "fasp" and f is an object of class "fv". Then eval.fasp(X + f) will add the function f to the functions in each cell of the array X, and return the resulting array of functions.

In general, expr can be any expression involving (a) the *names* of objects of class "fasp" or "fv", (b) scalar constants, and (c) functions which are vectorised. See the Examples.

First eval.fasp determines which of the *variable names* in the expression expr refer to objects of class "fasp". The expression is then evaluated for each cell of the array using eval.fv.

The expression expr must be vectorised. There must be at least one object of class "fasp" in the expression. All such objects must be compatible.

Value

Another object of class "fasp".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

fasp.object, Kest

Examples

```
# manipulating the K function
K <- alltypes(amacrine, "K")

# expressions involving a fasp object
eval.fasp(K + 3)
L <- eval.fasp(sqrt(K/pi))

# expression involving two fasp objects
D <- eval.fasp(K - L)

# subtracting the unmarked K function from the cross-type K functions
K0 <- Kest(unmark(amacrine))
DK <- eval.fasp(K - K0)

## Use of 'envir'
S <- eval.fasp(1-G, list(G=alltypes(amacrine, "G")))
```

`eval.fv` *Evaluate Expression Involving Functions*

Description

Evaluates any expression involving one or more function value (fv) objects, and returns another object of the same kind.

Usage

```
eval.fv(expr, envir, dotonly=TRUE, equiv=NULL, relabel=TRUE)
```

Arguments

`expr`	An expression.
`envir`	Optional. The environment in which to evaluate the expression, or a named list containing `"fv"` objects to be used in the expression.
`dotonly`	Logical. See Details.
`equiv`	Mapping between column names of different objects that are deemed to be equivalent. See Details.
`relabel`	Logical value indicating whether to compute appropriate labels for the resulting function. This should normally be `TRUE` (the default). See Details.

Details

This is a wrapper to make it easier to perform pointwise calculations with the summary functions used in spatial statistics.

An object of class `"fv"` is essentially a data frame containing several different statistical estimates of the same function. Such objects are returned by `Kest` and its relatives.

For example, suppose `X` is an object of class `"fv"` containing several different estimates of the Ripley's K function $K(r)$, evaluated at a sequence of values of r. Then `eval.fv(X+3)` effectively adds 3 to each function estimate in `X`, and returns the resulting object.

Suppose X and Y are two objects of class "fv" which are compatible (in particular they have the same vector of r values). Then eval.im(X + Y) will add the corresponding function values in X and Y, and return the resulting function.

In general, expr can be any expression involving (a) the *names* of objects of class "fv", (b) scalar constants, and (c) functions which are vectorised. See the Examples.

First eval.fv determines which of the *variable names* in the expression expr refer to objects of class "fv". Each such name is replaced by a vector containing the function values. The expression is then evaluated. The result should be a vector; it is taken as the new vector of function values.

The expression expr must be vectorised. There must be at least one object of class "fv" in the expression. If the objects are not compatible, they will be made compatible by harmonise.fv.

If dotonly=TRUE (the default), the expression will be evaluated only for those columns of an "fv" object that contain values of the function itself (rather than values of the derivative of the function, the hazard rate, etc). If dotonly=FALSE, the expression will be evaluated for all columns.

For example the result of Fest includes several columns containing estimates of the empty space function $F(r)$, but also includes an estimate of the *hazard* $h(r)$ of $F(r)$. Transformations that are valid for F may not be valid for h. Accordingly, h would normally be omitted from the calculation.

The columns of an object x that represent the function itself are identified by its "dot" names, fvnames(x, "."). They are the columns normally plotted by plot.fv and identified by the symbol "." in plot formulas in plot.fv.

The argument equiv can be used to specify that two different column names in different function objects are mathematically equivalent or cognate. It should be a list of name=value pairs, or a named vector of character strings, indicating the pairing of equivalent names. (Without this argument, these columns would be discarded.) See the Examples.

The argument relabel should normally be TRUE (the default). It determines whether to compute appropriate mathematical labels and descriptions for the resulting function object (used when the object is printed or plotted). If relabel=FALSE then this does not occur, and the mathematical labels and descriptions in the result are taken from the function object that appears first in the expression. This reduces computation time slightly (for advanced use only).

Value

Another object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

fv.object, Kest

Examples

```
# manipulating the K function
X <- rpoispp(42)
Ks <- Kest(X)

eval.fv(Ks + 3)
Ls <- eval.fv(sqrt(Ks/pi))
```

```
# manipulating two K functions
Y <- rpoispp(20)
Kr <- Kest(Y)

Kdif <- eval.fv(Ks - Kr)
Z <- eval.fv(sqrt(Ks/pi) - sqrt(Kr/pi))

## Use of 'envir'
U <- eval.fv(sqrt(K), list(K=Kest(cells)))

## Use of 'equiv'
Fc <- Fest(cells)
Gc <- Gest(cells)
# Hanisch and Chiu-Stoyan estimators are cognate
Dc <- eval.fv(Fc - Gc, equiv=list(cs="han"))
```

`eval.im` *Evaluate Expression Involving Pixel Images*

Description

Evaluates any expression involving one or more pixel images, and returns a pixel image.

Usage

```
eval.im(expr, envir, harmonize=TRUE)
```

Arguments

`expr`	An expression.
`envir`	Optional. The environment in which to evaluate the expression, or a named list containing pixel images to be used in the expression.
`harmonize`	Logical. Whether to resolve inconsistencies between the pixel grids.

Details

This function is a wrapper to make it easier to perform pixel-by-pixel calculations in an image.

Pixel images in **spatstat** are represented by objects of class `"im"` (see `im.object`). These are essentially matrices of pixel values, with extra attributes recording the pixel dimensions, etc.

Suppose `X` is a pixel image. Then `eval.im(X+3)` will add 3 to the value of every pixel in `X`, and return the resulting pixel image.

Suppose `X` and `Y` are two pixel images with compatible dimensions: they have the same number of pixels, the same physical size of pixels, and the same bounding box. Then `eval.im(X + Y)` will add the corresponding pixel values in `X` and `Y`, and return the resulting pixel image.

In general, `expr` can be any expression in the R language involving (a) the *names* of pixel images, (b) scalar constants, and (c) functions which are vectorised. See the Examples.

First `eval.im` determines which of the *variable names* in the expression `expr` refer to pixel images. Each such name is replaced by a matrix containing the pixel values. The expression is then evaluated. The result should be a matrix; it is taken as the matrix of pixel values.

The expression `expr` must be vectorised. There must be at least one pixel image in the expression.

All images must have compatible dimensions. If harmonize=TRUE, images that have incompatible dimensions will be resampled so that they are compatible. If harmonize=FALSE, images that are incompatible will cause an error.

Value

An image object of class "im".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

as.im, compatible.im, harmonise.im, im.object

Examples

```
# test images
X <- as.im(function(x,y) { x^2 - y^2 }, unit.square())
Y <- as.im(function(x,y) { 3 * x + y }, unit.square())

eval.im(X + 3)
eval.im(X - Y)
eval.im(abs(X - Y))
Z <- eval.im(sin(X * pi) + Y)

## Use of 'envir'
W <- eval.im(sin(U), list(U=density(cells)))
```

---

eval.linim *Evaluate Expression Involving Pixel Images on Linear Network*

Description

Evaluates any expression involving one or more pixel images on a linear network, and returns a pixel image on the same linear network.

Usage

```
eval.linim(expr, envir, harmonize=TRUE)
```

Arguments

expr	An expression in the R language, involving the names of objects of class "linim".
envir	Optional. The environment in which to evaluate the expression.
harmonize	Logical. Whether to resolve inconsistencies between the pixel grids.

Details

This function a wrapper to make it easier to perform pixel-by-pixel calculations. It is one of several functions whose names begin with `eval` which work on objects of different types. This particular function is designed to work with objects of class `"linim"` which represent pixel images on a linear network.

Suppose `X` is a pixel image on a linear network (object of class `"linim"`. Then `eval.linim(X+3)` will add 3 to the value of every pixel in `X`, and return the resulting pixel image on the same linear network.

Suppose `X` and `Y` are two pixel images on the same linear network, with compatible pixel dimensions. Then `eval.linim(X + Y)` will add the corresponding pixel values in `X` and `Y`, and return the resulting pixel image on the same linear network.

In general, `expr` can be any expression in the R language involving (a) the *names* of pixel images, (b) scalar constants, and (c) functions which are vectorised. See the Examples.

First `eval.linim` determines which of the *variable names* in the expression `expr` refer to pixel images. Each such name is replaced by a matrix containing the pixel values. The expression is then evaluated. The result should be a matrix; it is taken as the matrix of pixel values.

The expression `expr` must be vectorised. There must be at least one linear pixel image in the expression.

All images must have compatible dimensions. If `harmonize=TRUE`, images that have incompatible dimensions will be resampled so that they are compatible. If `harmonize=FALSE`, images that are incompatible will cause an error.

Value

An image object of class `"linim"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`eval.im`, `linim`

Examples

```
M <- as.mask.psp(as.psp(simplenet))
Z <- as.im(function(x,y) {x-y}, W=M)
X <- linim(simplenet, Z)
X

Y <- linfun(function(x,y,seg,tp){y^2+x}, simplenet)
Y <- as.linim(Y)

eval.linim(X + 3)
eval.linim(X - Y)
eval.linim(abs(X - Y))
Z <- eval.linim(sin(X * pi) + Y)
```

ewcdf *Weighted Empirical Cumulative Distribution Function*

Description

Compute a weighted version of the empirical cumulative distribution function.

Usage

```
ewcdf(x, weights = rep(1/length(x), length(x)))
```

Arguments

`x`	Numeric vector of observations.
`weights`	Numeric vector of non-negative weights for `x`.

Details

This is a modification of the standard function `ecdf` allowing the observations `x` to have weights.

The weighted e.c.d.f. (empirical cumulative distribution function) `Fn` is defined so that, for any real number `y`, the value of `Fn(y)` is equal to the total weight of all entries of `x` that are less than or equal to `y`. That is `Fn(y) = sum(weights[x <= y])`.

Thus `Fn` is a step function which jumps at the values of `x`. The height of the jump at a point `y` is the total weight of all entries in `x` number of tied observations at that value. Missing values are ignored.

If `weights` is omitted, the default is equivalent to `ecdf(x)` except for the class membership.

The result of `ewcdf` is a function, of class `"ewcdf"`, inheriting from the classes `"ecdf"` and `"stepfun"`. The class `ewcdf` has methods for `print` and `quantile`. The inherited class `ecdf` has methods for `plot` and `summary`.

Value

A function, of class `"ewcdf"`, inheriting from `"ecdf"` and `"stepfun"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`ecdf`.

`quantile.ewcdf`

Examples

```
x <- rnorm(100)
w <- runif(100)
plot(ewcdf(x,w))
```

`exactMPLEstrauss` *Exact Maximum Pseudolikelihood Estimate for Stationary Strauss Process*

Description

Computes, to very high accuracy, the Maximum Pseudolikelihood Estimates of the parameters of a stationary Strauss point process.

Usage

```
exactMPLEstrauss(X, R, ngrid = 2048, plotit = FALSE, project=TRUE)
```

Arguments

`X`	Data to which the Strauss process will be fitted. A point pattern dataset (object of class `"ppp"`).
`R`	Interaction radius of the Strauss process. A non-negative number.
`ngrid`	Grid size for calculation of integrals. An integer, giving the number of grid points in the x and y directions.
`plotit`	Logical. If `TRUE`, the log pseudolikelihood is plotted on the current device.
`project`	Logical. If `TRUE` (the default), the parameter γ is constrained to lie in the interval $[0, 1]$. If `FALSE`, this constraint is not applied.

Details

This function is intended mainly for technical investigation of algorithm performance. Its practical use is quite limited.

It fits the stationary Strauss point process model to the point pattern dataset `X` by maximum pseudolikelihood (with the border edge correction) using an algorithm with very high accuracy. This algorithm is more accurate than the *default* behaviour of the model-fitting function `ppm` because the discretisation is much finer.

Ripley (1988) and Baddeley and Turner (2000) derived the log pseudolikelihood for the stationary Strauss process, and eliminated the parameter β, obtaining an exact formula for the partial log pseudolikelihood as a function of the interaction parameter γ only. The algorithm evaluates this expression to a high degree of accuracy, using numerical integration on a `ngrid * ngrid` lattice, uses `optim` to maximise the log pseudolikelihood with respect to γ, and finally recovers β.

The result is a vector of length 2, containing the fitted coefficients $\log\beta$ and $\log\gamma$. These values correspond to the entries that would be obtained with `coef(ppm(X, ~1, Strauss(R)))`. The fitted coefficients are typically accurate to within 10^{-6} as shown in Baddeley and Turner (2013).

Note however that (by default) `exactMPLEstrauss` constrains the parameter γ to lie in the interval $[0, 1]$ in which the point process is well defined (Kelly and Ripley, 1976) whereas `ppm` does not constrain the value of γ (by default). This behaviour is controlled by the argument `project` to `ppm` and `exactMPLEstrauss`. The default for `ppm` is `project=FALSE`, while the default for `exactMPLEstrauss` is `project=TRUE`.

Value

Vector of length 2.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Baddeley, A. and Turner, R. (2013) Bias correction for parameter estimates of spatial point process models. *Journal of Statistical Computation and Simulation* **2012**. doi: 10.1080/00949655.2012.755976

Kelly, F.P. and Ripley, B.D. (1976) On Strauss's model for clustering. *Biometrika* **63**, 357–360.

Ripley, B.D. (1988) *Statistical inference for spatial processes*. Cambridge University Press.

See Also

ppm

Examples

```
if(interactive()) {
   exactMPLEstrauss(cells, 0.1)
   coef(ppm(cells, ~1, Strauss(0.1)))
   coef(ppm(cells, ~1, Strauss(0.1), nd=128))
   exactMPLEstrauss(redwood, 0.04)
   exactMPLEstrauss(redwood, 0.04, project=FALSE)
   coef(ppm(redwood, ~1, Strauss(0.04)))
}
```

---

expand.owin *Apply Expansion Rule*

Description

Applies an expansion rule to a window.

Usage

```
expand.owin(W, ...)
```

Arguments

W	A window.
...	Arguments passed to rmhexpand to determine an expansion rule.

Details

The argument W should be a window (an object of class "owin").

This command applies the expansion rule specified by the arguments ... to the window W, yielding another window.

The arguments ... are passed to rmhexpand to determine the expansion rule.

For other transformations of the scale, location and orientation of a window, see shift, affine and rotate.

Value

A window (object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rmhexpand about expansion rules.

shift, rotate, affine for other types of manipulation.

Examples

```
expand.owin(square(1), 9)
expand.owin(square(1), distance=0.5)
expand.owin(letterR, length=2)
expand.owin(letterR, distance=0.1)
```

Extract.anylist *Extract or Replace Subset of a List of Things*

Description

Extract or replace a subset of a list of things.

Usage

```
  ## S3 method for class 'anylist'
x[i, ...]

  ## S3 replacement method for class 'anylist'
x[i] <- value
```

Arguments

x	An object of class "anylist" representing a list of things.
i	Subset index. Any valid subset index in the usual R sense.
value	Replacement value for the subset.
...	Ignored.

Details

These are the methods for extracting and replacing subsets for the class "anylist".

The argument x should be an object of class "anylist" representing a list of things. See anylist.

The method replaces a designated subset of x, and returns an object of class "anylist".

Value

Another object of class "anylist".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

anylist, plot.anylist, summary.anylist

Examples

```
x <- anylist(A=runif(10), B=runif(10), C=runif(10))
x[1] <- list(A=rnorm(10))
```

Extract.fasp *Extract Subset of Function Array*

Description

Extract a subset of a function array (an object of class "fasp").

Usage

```
  ## S3 method for class 'fasp'
x[I, J, drop=TRUE,...]
```

Arguments

x	A function array. An object of class "fasp".
I	any valid expression for a subset of the row indices of the array.
J	any valid expression for a subset of the column indices of the array.
drop	Logical. When the selected subset consists of only one cell of the array, if drop=FALSE the result is still returned as a 1×1 array of functions (class "fasp") while if drop=TRUE it is returned as a function (class "fv").
...	Ignored.

Details

A function array can be regarded as a matrix whose entries are functions. See fasp.object for an explanation of function arrays.

This routine extracts a sub-array according to the usual conventions for matrix indexing.

Value

A function array (of class "fasp"). Exceptionally, if the array has only one cell, and if drop=TRUE, then the result is a function value table (class "fv").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

fasp.object

Examples

```
# Lansing woods data - multitype points with 6 types
woods <- lansing

# compute 6 x 6 array of all cross-type K functions
a <- alltypes(woods, "K")

# extract first three marks only
b <- a[1:3,1:3]
## Not run: plot(b)
# subset of array pertaining to hickories
h <- a[levels(marks(woods)) == "hickory", ]
## Not run: plot(h)
```

Extract.fv *Extract or Replace Subset of Function Values*

Description

Extract or replace a subset of an object of class "fv".

Usage

```
  ## S3 method for class 'fv'
x[i, j, ..., drop=FALSE]

  ## S3 replacement method for class 'fv'
x[i, j] <- value

  ## S3 replacement method for class 'fv'
x$name <- value
```

Arguments

x	a function value object, of class "fv" (see fv.object). Essentially a data frame.
i	any appropriate subset index. Selects a subset of the rows of the data frame, i.e. a subset of the domain of the function(s) represented by x.

j	any appropriate subset index for the columns of the data frame. Selects some of the functions present in x.
name	the name of a column of the data frame.
...	Ignored.
drop	Logical. If TRUE, the result is a data frame or vector containing the selected rows and columns of data. If FALSE (the default), the result is another object of class "fv".
value	Replacement value for the column or columns selected by name or j.

Details

These functions extract a designated subset of an object of class "fv", or replace the designated subset with other data, or delete the designated subset.

The subset is specified by the row index i and column index j, or by the column name name. Either i or j may be missing, or both may be missing.

The function [.fv is a method for the generic operator [for the class "fv". It extracts the designated subset of x, and returns it as another object of class "fv" (if drop=FALSE) or as a data frame or vector (if drop=TRUE).

The function [<-.fv is a method for the generic operator [<- for the class "fv". If value is NULL, the designated subset of x will be deleted from x. Otherwise, the designated subset of x will be replaced by the data contained in value. The return value is the modified object x.

The function $<-.fv is a method for the generic operator $<- for the class "fv". If value is NULL, the designated column of x will be deleted from x. Otherwise, the designated column of x will be replaced by the data contained in value. The return value is the modified object x.

Value

The result of [.fv with drop=TRUE is a data frame or vector.

Otherwise, the result is another object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

fv.object

Examples

```
K <- Kest(cells)

# discard the estimates of K(r) for r  > 0.1
Ksub <- K[K$r <= 0.1, ]

# extract the border method estimates
bor <- K[ , "border", drop=TRUE]
# or equivalently
bor <- K$border

# remove the border-method estimates
K$border <- NULL
K
```

`Extract.hyperframe` *Extract or Replace Subset of Hyperframe*

Description

Extract or replace a subset of a hyperframe.

Usage

```
  ## S3 method for class 'hyperframe'
x[i, j, drop, strip=drop, ...]
  ## S3 replacement method for class 'hyperframe'
x[i, j] <- value
  ## S3 method for class 'hyperframe'
x$name
  ## S3 replacement method for class 'hyperframe'
x$name <- value
```

Arguments

`x`	A hyperframe (object of class "hyperframe").
`i,j`	Row and column indices.
`drop,strip`	Logical values indicating what to do when the hyperframe has only one row or column. See Details.
`...`	Ignored.
`name`	Name of a column of the hyperframe.
`value`	Replacement value for the subset. A hyperframe or (if the subset is a single column) a list or an atomic vector.

Details

These functions extract a designated subset of a hyperframe, or replace the designated subset with another hyperframe.

The function `[.hyperframe` is a method for the subset operator `[` for the class "hyperframe". It extracts the subset of `x` specified by the row index `i` and column index `j`.

The argument `drop` determines whether the array structure will be discarded if possible. The argument `strip` determines whether the list structure in a row or column or cell will be discarded if possible. If `drop=FALSE` (the default), the return value is always a hyperframe or data frame. If `drop=TRUE`, and if the selected subset has only one row, or only one column, or both, then

- if `strip=FALSE`, the result is a list, with one entry for each array cell that was selected.
- if `strip=TRUE`,
  - if the subset has one row containing several columns, the result is a list or (if possible) an atomic vector;
  - if the subset has one column containing several rows, the result is a list or (if possible) an atomic vector;
  - if the subset has exactly one row and exactly one column, the result is the object (or atomic value) contained in this row and column.

The function `[<-.hyperframe` is a method for the subset replacement operator `[<-` for the class `"hyperframe"`. It replaces the designated subset with the hyperframe `value`. The subset of `x` to be replaced is designated by the arguments `i` and `j` as above. The replacement `value` should be a hyperframe with the appropriate dimensions, or (if the specified subset is a single column) a list of the appropriate length.

The function `$.hyperframe` is a method for `$` for hyperframes. It extracts the relevant column of the hyperframe. The result is always a list (i.e. equivalent to using `[.hyperframe` with `strip=FALSE`).

The function `$<-.hyperframe` is a method for `$<-` for hyperframes. It replaces the relevant column of the hyperframe. The replacement value should be a list of the appropriate length.

Value

A hyperframe (of class `"hyperframe"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`hyperframe`

Examples

```
h <- hyperframe(X=list(square(1), square(2)), Y=list(sin, cos))
h
h[1, ]
h[1, ,drop=TRUE]
h[ , 1]
h[ , 1, drop=TRUE]
h[1,1]
h[1,1,drop=TRUE]
h[1,1,drop=TRUE,strip=FALSE]
h[1,1] <- list(square(3))
# extract column
h$X
# replace existing column
h$Y <- list(cells, cells)
# add new column
h$Z <- list(cells, cells)
```

`Extract.im` *Extract Subset of Image*

Description

Extract a subset or subregion of a pixel image.

Usage

```
## S3 method for class 'im'
x[i, j, ..., drop=TRUE, tight=FALSE,
                                     raster=NULL, rescue=is.owin(i)]
```

Arguments

`x`	A two-dimensional pixel image. An object of class `"im"`.
`i`	Object defining the subregion or subset to be extracted. Either a spatial window (an object of class `"owin"`), or a pixel image with logical values, or a linear network (object of class `"linnet"`) or a point pattern (an object of class `"ppp"`), or any type of index that applies to a matrix, or something that can be converted to a point pattern by `as.ppp` (using the window of `x`).
`j`	An integer or logical vector serving as the column index if matrix indexing is being used. Ignored if `i` is a spatial object.
`...`	Ignored.
`drop`	Logical value. Locations in `w` that lie outside the spatial domain of the image `x` return a pixel value of `NA` if `drop=FALSE`, and are omitted if `drop=TRUE`.
`tight`	Logical value. If `tight=TRUE`, and if the result of the subset operation is an image, the image will be trimmed to the smallest possible rectangle.
`raster`	Optional. An object of class `"owin"` or `"im"` determining a pixel grid.
`rescue`	Logical value indicating whether rectangular blocks of data should always be returned as pixel images.

Details

This function extracts a subset of the pixel values in a pixel image. (To reassign the pixel values, see `[<-.im`).

The image `x` must be an object of class `"im"` representing a pixel image defined inside a rectangle in two-dimensional space (see `im.object`).

The subset to be extracted is determined by the arguments `i,j` according to the following rules (which are checked in this order):

1. `i` is a spatial object such as a window, a pixel image with logical values, a linear network, or a point pattern; or
2. `i,j` are indices for the matrix `as.matrix(x)`; or
3. `i` can be converted to a point pattern by `as.ppp(i, W=Window(x))`, and `i` is not a matrix.

If `i` is a spatial window (an object of class `"owin"`), the values of the image inside this window are extracted (after first clipping the window to the spatial domain of the image if necessary).

If `i` is a linear network (object of class `"linnet"`), the values of the image on this network are extracted.

If `i` is a pixel image with logical values, it is interpreted as a spatial window (with `TRUE` values inside the window and `FALSE` outside).

If `i` is a point pattern (an object of class `"ppp"`), then the values of the pixel image at the points of this pattern are extracted. This is a simple way to read the pixel values at a given spatial location.

At locations outside the spatial domain of the image, the pixel value is undefined, and is taken to be `NA`. The logical argument `drop` determines whether such `NA` values will be returned or omitted. It also influences the format of the return value.

If `i` is a point pattern (or something that can be converted to a point pattern), then `X[i, drop=FALSE]` is a numeric vector containing the pixel values at each of the points of the pattern. Its length is equal to the number of points in the pattern `i`. It may contain `NA`s corresponding to points which lie outside the spatial domain of the image `x`. By contrast, `X[i]` or `X[i, drop=TRUE]` contains only those pixel values which are not `NA`. It may be shorter.

If i is a spatial window then X[i, drop=FALSE] is another pixel image of the same dimensions as X obtained by setting all pixels outside the window i to have value NA. When the result is displayed by plot.im the effect is that the pixel image x is clipped to the window i.

If i is a linear network (object of class "linnet") then X[i, drop=FALSE] is another pixel image of the same dimensions as X obtained by restricting the pixel image X to the linear network. The result also belongs to the class "linim" (pixel image on a linear network).

If i is a spatial window then X[i, drop=TRUE] is either:

- a numeric vector containing the pixel values for all pixels that lie inside the window i. This happens if i is *not* a rectangle (i.e. i$type != "rectangle") or if rescue=FALSE.
- a pixel image. This happens only if i is a rectangle (i$type = "rectangle") and rescue=TRUE (the default).

If the optional argument raster is given, then it should be a binary image mask or a pixel image. Then x will first be converted to an image defined on the pixel grid implied by raster, before the subset operation is carried out. In particular, x[i, raster=i, drop=FALSE] will return an image defined on the same pixel array as the object i.

If i does not satisfy any of the conditions above, then the algorithm attempts to interpret i and j as indices for the matrix as.matrix(x). Either i or j may be missing or blank. The result is usually a vector or matrix of pixel values. Exceptionally the result is a pixel image if i,j determines a rectangular subset of the pixel grid, and if the user specifies rescue=TRUE.

Finally, if none of the above conditions is met, the object i may also be a data frame or list of x,y coordinates which will be converted to a point pattern, taking the observation window to be Window(x). Then the pixel values at these points will be extracted as a vector.

Value

Either a pixel image or a vector of pixel values. See Details.

Warnings

If you have a 2-column matrix containing the x, y coordinates of point locations, then to prevent this being interpreted as an array index, you should convert it to a data.frame or to a point pattern.

If W is a window or a pixel image, then x[W, drop=FALSE] will return an image defined on the same pixel array as the original image x. If you want to obtain an image whose pixel dimensions agree with those of W, use the raster argument, x[W, raster=W, drop=FALSE].

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

im.object, [<-.im, ppp.object, as.ppp, owin.object, plot.im

Examples

```
# make up an image
X <- setcov(unit.square())
plot(X)

# a rectangular subset
```

```
W <- owin(c(0,0.5),c(0.2,0.8))
Y <- X[W]
plot(Y)

# a polygonal subset
R <- affine(letterR, diag(c(1,1)/2), c(-2,-0.7))
plot(X[R, drop=FALSE])
plot(X[R, drop=FALSE, tight=TRUE])

# a point pattern
P <- rpoispp(20)
Y <- X[P]

# look up a specified location
X[list(x=0.1,y=0.2)]

# 10 x 10 pixel array
X <- as.im(function(x,y) { x + y }, owin(c(-1,1),c(-1,1)), dimyx=10)
# 100 x 100
W <- as.mask(disc(1, c(0,0)), dimyx=100)
# 10 x 10 raster
X[W,drop=FALSE]
# 100 x 100 raster
X[W, raster=W, drop=FALSE]
```

`Extract.influence.ppm` *Extract Subset of Influence Object*

Description

Extract a subset of an influence object, or extract the influence values at specified locations.

Usage

```
## S3 method for class 'influence.ppm'
x[i, ...]
```

Arguments

`x`	A influence object (of class "`influence.ppm`") computed by `influence.ppm`.
`i`	Subset index (passed to `[.ppp`). Either a spatial window (object of class "`owin`") or an integer index.
`...`	Ignored.

Details

An object of class "`influence.ppm`" contains the values of the likelihood influence for a point process model, computed by `influence.ppm`. This is effectively a marked point pattern obtained by marking each of the original data points with its likelihood influence.

This function extracts a designated subset of the influence values, either as another influence object, or as a vector of numeric values.

The function `[.influence.ppm` is a method for `[` for the class `"influence.ppm"`. The argument `i` should be an index applicable to a point pattern. It may be either a spatial window (object of class `"owin"`) or a sequence index. The result will be another influence object (of class `influence.ppm`).

To extract the influence values as a numeric vector, use `marks(as.ppp(x))`.

Value

Another object of class `"influence.ppm"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`influence.ppm`.

Examples

```
fit <- ppm(cells, ~x)
infl <- influence(fit)
b <- owin(c(0.1, 0.3), c(0.2, 0.4))
infl[b]
infl[1:5]
marks(as.ppp(infl))[1:3]
```

Extract.layered — *Extract or Replace Subset of a Layered Object*

Description

Extract or replace some or all of the layers of a layered object, or extract a spatial subset of each layer.

Usage

```
  ## S3 method for class 'layered'
x[i, j, drop=FALSE, ...]

  ## S3 replacement method for class 'layered'
x[i] <- value

  ## S3 replacement method for class 'layered'
x[[i]] <- value
```

Arguments

- `x` — A layered object (class "layered").
- `i` — Subset index for the list of layers. A logical vector, integer vector or character vector specifying which layers are to be extracted or replaced.
- `j` — Subset index to be applied to the data in each layer. Typically a spatial window (class "owin").
- `drop` — Logical. If `i` specifies only a single layer and `drop=TRUE`, then the contents of this layer will be returned.
- `...` — Additional arguments, passed to other subset methods if the subset index is a window.
- `value` — List of objects which shall replace the designated subset, or an object which shall replace the designated element.

Details

A layered object represents data that should be plotted in successive layers, for example, a background and a foreground. See `layered`.

The function `[.layered` extracts a designated subset of a layered object. It is a method for `[` for the class "layered".

The functions `[<-.layered` and `[[<-.layered` replace a designated subset or designated entry of the object by new values. They are methods for `[<-` and `[[<-` for the "layered" class.

The index `i` specifies which layers will be retained. It should be a valid subset index for the list of layers.

The index `j` will be applied to each layer. It is typically a spatial window (class "owin") so that each of the layers will be restricted to the same spatial region. Alternatively `j` may be any subset index which is permissible for the "[" method for each of the layers.

Value

Usually an object of class "layered".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`layered`

Examples

```
D <- distmap(cells)
L <- layered(D, cells,
             plotargs=list(list(ribbon=FALSE), list(pch=16)))

L[-2]
L[, square(0.5)]

L[[3]] <- japanesepines
L
```

Extract.leverage.ppm *Extract Subset of Leverage Object*

Description

Extract a subset of a leverage map, or extract the leverage values at specified locations.

Usage

```
## S3 method for class 'leverage.ppm'
x[i, ..., update=TRUE]
```

Arguments

x	A leverage object (of class "leverage.ppm") computed by leverage.ppm.
i	Subset index (passed to [.im). Either a spatial window (object of class "owin") or a spatial point pattern (object of class "ppp").
...	Further arguments passed to [.im, especially the argument drop.
update	Logical value indicating whether to update the internally-stored value of the mean leverage, by averaging over the specified subset.

Details

An object of class "leverage.ppm" contains the values of the leverage function for a point process model, computed by leverage.ppm.

This function extracts a designated subset of the leverage values, either as another leverage object, or as a vector of numeric values.

The function [.leverage.ppm is a method for [for the class "leverage.ppm". The argument i should be either

- a spatial window (object of class "owin") determining a region where the leverage map is required. The result will typically be another leverage map (object of class leverage.ppm).
- a spatial point pattern (object of class "ppp") specifying locations at which the leverage values are required. The result will be a numeric vector.

The subset operator for images, [.im, is applied to the leverage map. If this yields a pixel image, then the result of [.leverage.ppm is another leverage object. Otherwise, a vector containing the numeric values of leverage is returned.

Value

Another object of class "leverage.ppm", or a vector of numeric values of leverage.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

leverage.ppm.

Examples

```
fit <- ppm(cells ~x)
lev <- leverage(fit)
b <- owin(c(0.1, 0.3), c(0.2, 0.4))
lev[b]
lev[cells]
```

Extract.linim   *Extract Subset of Pixel Image on Linear Network*

Description

Extract a subset of a pixel image on a linear network.

Usage

```
  ## S3 method for class 'linim'
x[i, ..., drop=TRUE]
```

Arguments

x	A pixel image on a linear network (object of class "linim").
i	Spatial window defining the subregion. Either a spatial window (an object of class "owin"), or a logical-valued pixel image, or any type of index that applies to a matrix, or a point pattern (an object of class "lpp" or "ppp"), or something that can be converted to a point pattern by as.lpp (using the network on which x is defined).
...	Additional arguments passed to [.im.
drop	Logical value indicating whether NA values should be omitted from the result.

Details

This function is a method for the subset operator "[" for pixel images on linear networks (objects of class "linim").

The pixel image x will be restricted to the domain specified by i.

Pixels outside the domain of x are assigned the value NA; if drop=TRUE (the default) such NA values are deleted from the result; if drop=FALSE, then NA values are retained.

If i is a window (or a logical-valued pixel image) then x[i] is another pixel image of class "linim", representing the restriction of x to the spatial domain specified by i.

If i is a point pattern, then x[i] is the vector of pixel values of x at the locations specified by i.

Value

Another pixel image on a linear network (object of class "linim") or a vector of pixel values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Examples

```
M <- as.mask.psp(as.psp(simplenet))
Z <- as.im(function(x,y){x}, W=M)
Y <- linim(simplenet, Z)
X <- runiflpp(4, simplenet)
Y[X]
Y[square(c(0.3, 0.6))]
```

Extract.linnet *Extract Subset of Linear Network*

Description

Extract a subset of a linear network.

Usage

```
## S3 method for class 'linnet'
x[i, ..., snip=TRUE]
```

Arguments

x	A linear network (object of class "linnet").
i	Spatial window defining the subregion. An object of class "owin".
snip	Logical. If TRUE (the default), segments of x which cross the boundary of i will be cut by the boundary. If FALSE, these segments will be deleted.
...	Ignored.

Details

This function computes the intersection between the linear network x and the domain specified by i.

This function is a method for the subset operator "[" for linear networks (objects of class "linnet"). It is provided mainly for completeness.

The index i should be a window.

The argument snip specifies what to do with segments of x which cross the boundary of i. If snip=FALSE, such segments are simply deleted. If snip=TRUE (the default), such segments are cut into pieces by the boundary of i, and those pieces which lie inside the window i are included in the resulting network.

Value

Another linear network (object of class "linnet").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz>, Ege Rubak <rubak@math.aau.dk> and Suman Rakshit.

Examples

```
p <- par(mfrow=c(1,2), mar=0.2+c(0,0,1,0))
B <- owin(c(0.1,0.7),c(0.19,0.6))

plot(simplenet, main="x[w, snip=TRUE]")
plot(simplenet[B], add=TRUE, col="green", lwd=3)
plot(B, add=TRUE, border="red", lty=3)

plot(simplenet, main="x[w, snip=FALSE]")
plot(simplenet[B, snip=FALSE], add=TRUE, col="green", lwd=3)
plot(B, add=TRUE, border="red", lty=3)

par(p)
```

Extract.listof   *Extract or Replace Subset of a List of Things*

Description

Replace a subset of a list of things.

Usage

```
## S3 replacement method for class 'listof'
x[i] <- value
```

Arguments

x	An object of class "listof" representing a list of things which all belong to one class.
i	Subset index. Any valid subset index in the usual R sense.
value	Replacement value for the subset.

Details

This is a subset replacement method for the class "listof".

The argument x should be an object of class "listof" representing a list of things that all belong to one class.

The method replaces a designated subset of x, and returns an object of class "listof".

Value

Another object of class "listof".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

plot.listof, summary.listof

Examples

```
x <- list(A=runif(10), B=runif(10), C=runif(10))
class(x) <- c("listof", class(x))
x[1] <- list(A=rnorm(10))
```

`Extract.lpp` *Extract Subset of Point Pattern on Linear Network*

Description

Extract a subset of a point pattern on a linear network.

Usage

```
  ## S3 method for class 'lpp'
x[i, j, drop=FALSE, ..., snip=TRUE]
```

Arguments

x	A point pattern on a linear network (object of class "lpp").
i	Subset index. A valid subset index in the usual R sense, indicating which points should be retained.
j	Spatial window (object of class "owin") delineating the region that should be retained.
drop	Logical value indicating whether to remove unused levels of the marks, if the marks are a factor.
snip	Logical. If TRUE (the default), segments of the network which cross the boundary of the window j will be cut by the boundary. If FALSE, these segments will be deleted.
...	Ignored.

Details

This function extracts a designated subset of a point pattern on a linear network.

The function `[.lpp` is a method for `[` for the class "lpp". It extracts a designated subset of a point pattern. The argument `i` should be a subset index in the usual R sense: either a numeric vector of positive indices (identifying the points to be retained), a numeric vector of negative indices (identifying the points to be deleted) or a logical vector of length equal to the number of points in the point pattern `x`. In the latter case, the points `(x$x[i], x$y[i])` for which `subset[i]=TRUE` will be retained, and the others will be deleted.

The argument `j`, if present, should be a spatial window. The pattern inside the region will be retained. *Line segments that cross the boundary of the window are deleted* in the current implementation.

The argument `drop` determines whether to remove unused levels of a factor, if the point pattern is multitype (i.e. the marks are a factor) or if the marks are a data frame or hyperframe in which some of the columns are factors.

The argument `snip` specifies what to do with segments of the network which cross the boundary of the window `j`. If `snip=FALSE`, such segments are simply deleted. If `snip=TRUE` (the default), such segments are cut into pieces by the boundary of `j`, and those pieces which lie inside the window `ji` are included in the resulting network.

Use `unmark` to remove all the marks in a marked point pattern, and `subset.lpp` to remove only some columns of marks.

Value

A point pattern on a linear network (of class `"lpp"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`lpp`, `subset.lpp`

Examples

```
# Chicago crimes data - remove cases of assault
chicago[marks(chicago) != "assault"]
# equivalent to subset(chicago, select=-assault)

# spatial window subset
B <- owin(c(350, 700), c(600, 1000))
plot(chicago)
plot(B, add=TRUE, lty=2, border="red", lwd=3)
op <- par(mfrow=c(1,2), mar=0.6+c(0,0,1,0))
plot(B, main="chicago[B, snip=FALSE]", lty=3, border="red")
plot(chicago[, B, snip=FALSE], add=TRUE)
plot(B, main="chicago[B, snip=TRUE]", lty=3, border="red")
plot(chicago[, B, snip=TRUE], add=TRUE)
par(op)
```

`Extract.msr` *Extract Subset of Signed or Vector Measure*

Description

Extract a subset of a signed measure or vector-valued measure.

Usage

```
## S3 method for class 'msr'
x[i, j, ...]
```

Arguments

`x`	A signed or vector measure. An object of class `"msr"` (see `msr`).
`i`	Object defining the subregion or subset to be extracted. Either a spatial window (an object of class `"owin"`), or a pixel image with logical values, or any type of index that applies to a matrix.

j Subset index selecting the vector coordinates to be extracted, if x is a vector-valued measure.

... Ignored.

Details

This operator extracts a subset of the data which determines the signed measure or vector-valued measure x. The result is another measure.

Value

An object of class "msr".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

msr

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X ~x+y)
rp <- residuals(fit, type="pearson")
rs <- residuals(fit, type="score")

rp[square(0.5)]
rs[ , 2:3]
```

Extract.owin *Extract Subset of Window*

Description

Extract a subset of a window.

Usage

```
  ## S3 method for class 'owin'
x[i, ...]
```

Arguments

x A spatial window (object of class "owin").

i Object defining the subregion. Either a spatial window, or a pixel image with logical values.

... Ignored.

Details

This function computes the intersection between the window x and the domain specified by i, using intersect.owin.

This function is a method for the subset operator "[" for spatial windows (objects of class "owin"). It is provided mainly for completeness.

The index i may be either a window, or a pixel image with logical values (the TRUE values of the image specify the spatial domain).

Value

Another spatial window (object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

intersect.owin

Examples

```
W <- owin(c(2.5, 3.2), c(1.4, 2.9))
plot(letterR)
plot(letterR[W], add=TRUE, col="red")
```

Extract.ppp *Extract or Replace Subset of Point Pattern*

Description

Extract or replace a subset of a point pattern. Extraction of a subset has the effect of thinning the points and/or trimming the window.

Usage

```
  ## S3 method for class 'ppp'
x[i, j, drop=FALSE, ..., clip=FALSE]
  ## S3 replacement method for class 'ppp'
x[i, j] <- value
```

Arguments

x	A two-dimensional point pattern. An object of class "ppp".
i	Subset index. Either a valid subset index in the usual R sense, indicating which points should be retained, or a window (an object of class "owin") delineating a subset of the original observation window, or a pixel image with logical values defining a subset of the original observation window.
value	Replacement value for the subset. A point pattern.
j	Redundant. Included for backward compatibility.

drop — Logical value indicating whether to remove unused levels of the marks, if the marks are a factor.

clip — Logical value indicating how to form the window of the resulting point pattern, when i is a window. If clip=FALSE (the default), the result has window equal to i. If clip=TRUE, the resulting window is the intersection between the window of x and the window i.

... — Ignored. This argument is required for compatibility with the generic function.

Details

These functions extract a designated subset of a point pattern, or replace the designated subset with another point pattern.

The function [.ppp is a method for [for the class "ppp". It extracts a designated subset of a point pattern, either by "*thinning*" (retaining/deleting some points of a point pattern) or "*trimming*" (reducing the window of observation to a smaller subregion and retaining only those points which lie in the subregion) or both.

The pattern will be "thinned" if i is a subset index in the usual R sense: either a numeric vector of positive indices (identifying the points to be retained), a numeric vector of negative indices (identifying the points to be deleted) or a logical vector of length equal to the number of points in the point pattern x. In the latter case, the points (x$x[i], x$y[i]) for which subset[i]=TRUE will be retained, and the others will be deleted.

The pattern will be "trimmed" if i is an object of class "owin" specifying a window of observation. The points of x lying inside the new window i will be retained. Alternatively i may be a pixel image (object of class "im") with logical values; the pixels with the value TRUE will be interpreted as a window.

The argument drop determines whether to remove unused levels of a factor, if the point pattern is multitype (i.e. the marks are a factor) or if the marks are a data frame in which some of the columns are factors.

The function [<-.ppp is a method for [<- for the class "ppp". It replaces the designated subset with the point pattern value. The subset of x to be replaced is designated by the argument i as above.

The replacement point pattern value must lie inside the window of the original pattern x. The ordering of points in x will be preserved if the replacement pattern value has the same number of points as the subset to be replaced. Otherwise the ordering is unpredictable.

If the original pattern x has marks, then the replacement pattern value must also have marks, of the same type.

Use the function unmark to remove marks from a marked point pattern.

Use the function split.ppp to select those points in a marked point pattern which have a specified mark.

Value

A point pattern (of class "ppp").

Warnings

The function does not check whether i is a subset of Window(x). Nor does it check whether value lies inside Window(x).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

subset.ppp.

ppp.object, owin.object, unmark, split.ppp, cut.ppp

Examples

```
 # Longleaf pines data
 lon <- longleaf
 ## Not run:
 plot(lon)

## End(Not run)


 # adult trees defined to have diameter at least 30 cm
 longadult <- subset(lon, marks >= 30)
 ## Not run:
 plot(longadult)

## End(Not run)
 # note that the marks are still retained.
 # Use unmark(longadult) to remove the marks

 # New Zealand trees data
 ## Not run:
 plot(nztrees)          # plot shows a line of trees at the far right
 abline(v=148, lty=2)   # cut along this line

## End(Not run)
 nzw <- owin(c(0,148),c(0,95)) # the subwindow
 # trim dataset to this subwindow
 nzsub <- nztrees[nzw]
 ## Not run:
 plot(nzsub)

## End(Not run)

 # Redwood data
 ## Not run:
 plot(redwood)

## End(Not run)
 # Random thinning: delete 60% of data
 retain <- (runif(npoints(redwood)) < 0.4)
 thinred <- redwood[retain]
 ## Not run:
 plot(thinred)

## End(Not run)

 # Scramble 60% of data
```

```
X <- redwood
modif <- (runif(npoints(X)) < 0.6)
X[modif] <- runifpoint(ex=X[modif])

# Lansing woods data - multitype points
lan <- lansing


# Hickory trees
 hicks <- split(lansing)$hickory

# Trees in subwindow
 win <- owin(c(0.3, 0.6),c(0.2, 0.5))
 lsub <- lan[win]

# Scramble the locations of trees in subwindow, retaining their marks
 lan[win] <- runifpoint(ex=lsub) %mark% marks(lsub)

# Extract oaks only
oaknames <- c("redoak", "whiteoak", "blackoak")
oak <- lan[marks(lan) %in% oaknames, drop=TRUE]
oak <- subset(lan, marks %in% oaknames, drop=TRUE)

# To clip or not to clip
X <- runifpoint(25, letterR)
B <- owin(c(2.2, 3.9), c(2, 3.5))
opa <- par(mfrow=c(1,2))
plot(X, main="X[B]")
plot(X[B], border="red", cols="red", add=TRUE, show.all=TRUE, main="")
plot(X, main="X[B, clip=TRUE]")
plot(B, add=TRUE, lty=2)
plot(X[B, clip=TRUE], border="blue", cols="blue", add=TRUE,
     show.all=TRUE, main="")
par(opa)
```

Extract.ppx *Extract Subset of Multidimensional Point Pattern*

Description

Extract a subset of a multidimensional point pattern.

Usage

```
  ## S3 method for class 'ppx'
x[i, drop=FALSE, ...]
```

Arguments

x	A multidimensional point pattern (object of class "ppx").
i	Subset index. A valid subset index in the usual R sense, indicating which points should be retained; or a spatial domain of class "boxx" or "box3".
drop	Logical value indicating whether to remove unused levels of the marks, if the marks are a factor.
...	Ignored.

Details

This function extracts a designated subset of a multidimensional point pattern.

The function [.ppx is a method for [for the class "ppx". It extracts a designated subset of a point pattern. The argument i may be either

- a subset index in the usual R sense: either a numeric vector of positive indices (identifying the points to be retained), a numeric vector of negative indices (identifying the points to be deleted) or a logical vector of length equal to the number of points in the point pattern x. In the latter case, the points (x$x[i], x$y[i]) for which subset[i]=TRUE will be retained, and the others will be deleted.
- a spatial domain of class "boxx" or "box3". Points falling inside this region will be retained.

The argument drop determines whether to remove unused levels of a factor, if the point pattern is multitype (i.e. the marks are a factor) or if the marks are a data frame or hyperframe in which some of the columns are factors.

Use the function unmark to remove marks from a marked point pattern.

Value

A multidimensional point pattern (of class "ppx").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

ppx

Examples

```
df <- data.frame(x=runif(4),y=runif(4),z=runif(4))
X <- ppx(data=df, coord.type=c("s","s","t"))
X[-2]
```

Extract.psp *Extract Subset of Line Segment Pattern*

Description

Extract a subset of a line segment pattern.

Usage

```
  ## S3 method for class 'psp'
x[i, j, drop, ..., fragments=TRUE]
```

Arguments

x	A two-dimensional line segment pattern. An object of class "psp".
i	Subset index. Either a valid subset index in the usual R sense, indicating which segments should be retained, or a window (an object of class "owin") delineating a subset of the original observation window.
j	Redundant - included for backward compatibility.
drop	Ignored. Required for compatibility with generic function.
...	Ignored.
fragments	Logical value indicating whether to retain all pieces of line segments that intersect the new window (fragments=TRUE, the default) or to retain only those line segments that lie entirely inside the new window (fragments=FALSE).

Details

These functions extract a designated subset of a line segment pattern.

The function [.psp is a method for [for the class "psp". It extracts a designated subset of a line segment pattern, either by "*thinning*" (retaining/deleting some line segments of a line segment pattern) or "*trimming*" (reducing the window of observation to a smaller subregion and clipping the line segments to this boundary) or both.

The pattern will be "thinned" if subset is specified. The line segments designated by subset will be retained. Here subset can be a numeric vector of positive indices (identifying the line segments to be retained), a numeric vector of negative indices (identifying the line segments to be deleted) or a logical vector of length equal to the number of line segments in the line segment pattern x. In the latter case, the line segments for which subset[i]=TRUE will be retained, and the others will be deleted.

The pattern will be "trimmed" if window is specified. This should be an object of class owin specifying a window of observation to which the line segment pattern x will be trimmed. Line segments of x lying inside the new window will be retained unchanged. Line segments lying partially inside the new window and partially outside it will, by default, be clipped so that they lie entirely inside the window; but if fragments=FALSE, such segments will be removed.

Both "thinning" and "trimming" can be performed together.

Value

A line segment pattern (of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

psp.object, owin.object

Examples

```
  a <- psp(runif(20),runif(20),runif(20),runif(20), window=owin())
  plot(a)
# thinning
```

```
    id <- sample(c(TRUE, FALSE), 20, replace=TRUE)
    b <- a[id]
    plot(b, add=TRUE, lwd=3)
 # trimming
    plot(a)
    w <- owin(c(0.1,0.7), c(0.2, 0.8))
    b <- a[w]
    plot(b, add=TRUE, col="red", lwd=2)
    plot(w, add=TRUE)
    u <- a[w, fragments=FALSE]
    plot(u, add=TRUE, col="blue", lwd=3)
```

Extract.quad *Subset of Quadrature Scheme*

Description

Extract a subset of a quadrature scheme.

Usage

```
  ## S3 method for class 'quad'
x[...]
```

Arguments

x	A quadrature scheme (object of class "quad").
...	Arguments passed to [.ppp to determine the subset.

Details

This function extracts a designated subset of a quadrature scheme.

The function [.quad is a method for [for the class "quad". It extracts a designated subset of a quadrature scheme.

The subset to be extracted is determined by the arguments ... which are interpreted by [.ppp. Thus it is possible to take the subset consisting of all quadrature points that lie inside a given region, or a subset of quadrature points identified by numeric indices.

Value

A quadrature scheme (object of class "quad").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quad.object, [.ppp.

Examples

```
Q <- quadscheme(nztrees)
W <- owin(c(0,148),c(0,95)) # a subwindow
Q[W]
```

`Extract.solist` *Extract or Replace Subset of a List of Spatial Objects*

Description

Extract or replace some entries in a list of spatial objects, or extract a designated sub-region in each object.

Usage

```
  ## S3 method for class 'solist'
x[i, ...]

  ## S3 replacement method for class 'solist'
x[i] <- value
```

Arguments

`x`	An object of class "solist" representing a list of two-dimensional spatial objects.
`i`	Subset index. Any valid subset index for vectors in the usual R sense, or a window (object of class "owin").
`value`	Replacement value for the subset.
`...`	Ignored.

Details

These are methods for extracting and replacing subsets for the class "solist".

The argument `x` should be an object of class "solist" representing a list of two-dimensional spatial objects. See `solist`.

For the subset method, the subset index `i` can be either a vector index (specifying some elements of the list) or a spatial window (specifying a spatial sub-region).

For the replacement method, `i` must be a vector index: the designated elements will be replaced.

Value

Another object of the same class as `x`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

solist, plot.solist, summary.solist

Examples

```
x <- solist(japanesepines, cells, redwood)
x[2:3]
x[square(0.5)]
x[1] <- list(finpines)
```

Extract.splitppp *Extract or Replace Sub-Patterns*

Description

Extract or replace some of the sub-patterns in a split point pattern.

Usage

```
  ## S3 method for class 'splitppp'
x[...]
  ## S3 replacement method for class 'splitppp'
x[...] <- value
```

Arguments

x	An object of class "splitppp", representing a point pattern separated into a list of sub-patterns.
...	Subset index. Any valid subset index in the usual R sense.
value	Replacement value for the subset. A list of point patterns.

Details

These are subset methods for the class "splitppp".

The argument x should be an object of class "splitppp", representing a point pattern that has been separated into a list of sub-patterns. It is created by split.ppp.

The methods extract or replace a designated subset of the list x, and return an object of class "splitppp".

Value

Another object of class "splitppp".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

split.ppp, plot.splitppp, summary.splitppp

Examples

```
data(amacrine)  # multitype point pattern
y <- split(amacrine)
y[1]
y["off"]
y[1] <- list(runifpoint(42, Window(amacrine)))
```

`Extract.tess` *Extract or Replace Subset of Tessellation*

Description

Extract, change or delete a subset of the tiles of a tessellation, to make a new tessellation.

Usage

```
  ## S3 method for class 'tess'
x[i, ...]
  ## S3 replacement method for class 'tess'
x[i, ...] <- value
```

Arguments

`x`	A tessellation (object of class "tess").
`i`	Subset index for the tiles of the tessellation. Alternatively a window (object of class "owin").
`...`	One argument that specifies the subset to be extracted or changed. Any valid format for the subset index in a list.
`value`	Replacement value for the selected tiles of the tessellation. A list of windows (objects of class "owin") or NULL.

Details

A tessellation (object of class "tess", see tess) is effectively a list of tiles (spatial regions) that cover a spatial region. The subset operator `[.tess` extracts some of these tiles and forms a new tessellation, which of course covers a smaller region than the original.

For `[.tess` only, the subset index can also be a window (object of class "owin"). The tessellation `x` is then intersected with the window.

The replacement operator changes the selected tiles. The replacement `value` may be either `NULL` (which causes the selected tiles to be removed from `x`) or a list of the same length as the selected subset. The entries of `value` may be windows (objects of class "owin") or `NULL` to indicate that the corresponding tile should be deleted.

Generally it does not make sense to replace a tile in a tessellation with a completely different tile, because the tiles are expected to fit together. However this facility is sometimes useful for making small adjustments to polygonal tiles.

Value

A tessellation (object of class "tess").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

tess, tiles, intersect.tess.

Examples

```
A <- tess(xgrid=0:4, ygrid=0:3)
B <- A[c(1, 3, 7)]
E <- A[-1]
A[c(2, 5, 11)] <- NULL
```

F3est *Empty Space Function of a Three-Dimensional Point Pattern*

Description

Estimates the empty space function $F_3(r)$ from a three-dimensional point pattern.

Usage

```
F3est(X, ..., rmax = NULL, nrval = 128, vside = NULL,
               correction = c("rs", "km", "cs"),
               sphere = c("fudge", "ideal", "digital"))
```

Arguments

X	Three-dimensional point pattern (object of class "pp3").
...	Ignored.
rmax	Optional. Maximum value of argument r for which $F_3(r)$ will be estimated.
nrval	Optional. Number of values of r for which $F_3(r)$ will be estimated. A large value of nrval is required to avoid discretisation effects.
vside	Optional. Side length of the voxels in the discrete approximation.
correction	Optional. Character vector specifying the edge correction(s) to be applied. See Details.
sphere	Optional. Character string specifying how to calculate the theoretical value of $F_3(r)$ for a Poisson process. See Details.

Details

For a stationary point process Φ in three-dimensional space, the empty space function is

$$F_3(r) = P(d(0, \Phi) \le r)$$

where $d(0, \Phi)$ denotes the distance from a fixed origin 0 to the nearest point of Φ.

The three-dimensional point pattern `X` is assumed to be a partial realisation of a stationary point process Φ. The empty space function of Φ can then be estimated using techniques described in the References.

The box containing the point pattern is discretised into cubic voxels of side length `vside`. The distance function $d(u, \Phi)$ is computed for every voxel centre point u using a three-dimensional version of the distance transform algorithm (Borgefors, 1986). The empirical cumulative distribution function of these values, with appropriate edge corrections, is the estimate of $F_3(r)$.

The available edge corrections are:

"rs": the reduced sample (aka minus sampling, border correction) estimator (Baddeley et al, 1993)

"km": the three-dimensional version of the Kaplan-Meier estimator (Baddeley and Gill, 1997)

"cs": the three-dimensional generalisation of the Chiu-Stoyan or Hanisch estimator (Chiu and Stoyan, 1998).

Alternatively `correction="all"` selects all options.

The result includes a column `theo` giving the theoretical value of $F_3(r)$ for a uniform Poisson process (Complete Spatial Randomness). This value depends on the volume of the sphere of radius `r` measured in the discretised distance metric. The argument `sphere` determines how this will be calculated.

- If `sphere="ideal"` the calculation will use the volume of an ideal sphere of radius r namely $(4/3)\pi r^3$. This is not recommended because the theoretical values of $F_3(r)$ are inaccurate.
- If `sphere="fudge"` then the volume of the ideal sphere will be multiplied by 0.78, which gives the approximate volume of the sphere in the discretised distance metric.
- If `sphere="digital"` then the volume of the sphere in the discretised distance metric is computed exactly using another distance transform. This takes longer to compute, but is exact.

Value

A function value table (object of class `"fv"`) that can be plotted, printed or coerced to a data frame containing the function values.

Warnings

A small value of `vside` and a large value of `nrval` are required for reasonable accuracy.

The default value of `vside` ensures that the total number of voxels is `2^22` or about 4 million. To change the default number of voxels, see `spatstat.options("nvoxel")`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rana Moyeed.

References

Baddeley, A.J, Moyeed, R.A., Howard, C.V. and Boyde, A. Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42** (1993) 641–668.

Baddeley, A.J. and Gill, R.D. (1997) Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25**, 263–292.

Borgefors, G. (1986) Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34**, 344–371.

Chiu, S.N. and Stoyan, D. (1998) Estimators of distance distributions for spatial patterns. *Statistica Neerlandica* **52**, 239–246.

See Also

G3est, K3est, pcf3est.

Examples

```
X <- rpoispp3(42)
Z <- F3est(X)
if(interactive()) plot(Z)
```

---

fardist *Farthest Distance to Boundary of Window*

Description

Computes the farthest distance from each pixel, or each data point, to the boundary of the window.

Usage

```
  fardist(X, ...)

  ## S3 method for class 'owin'
fardist(X, ..., squared=FALSE)

  ## S3 method for class 'ppp'
fardist(X, ..., squared=FALSE)
```

Arguments

X	A spatial object such as a window or point pattern.
...	Arguments passed to as.mask to determine the pixel resolution, if required.
squared	Logical. If TRUE, the squared distances will be returned.

Details

The function `fardist` is generic, with methods for the classes `owin` and ppp.

For a window `W`, the command `fardist(W)` returns a pixel image in which the value at each pixel is the *largest* distance from that pixel to the boundary of `W`.

For a point pattern `X`, with window `W`, the command `fardist(X)` returns a numeric vector with one entry for each point of `X`, giving the largest distance from that data point to the boundary of `W`.

Value

For `fardist.owin`, a pixel image (object of class `"im"`).

For `fardist.ppp`, a numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
fardist(cells)

plot(FR <- fardist(letterR))
```

`fasp.object` *Function Arrays for Spatial Patterns*

Description

A class `"fasp"` to represent a "matrix" of functions, amenable to plotting as a matrix of plot panels.

Details

An object of this class is a convenient way of storing (and later plotting, editing, etc) a set of functions $f_{i,j}(r)$ of a real argument r, defined for each possible pair (i, j) of indices $1 \leq i, j \leq n$. We may think of this as a matrix or array of functions $f_{i,j}$.

Function arrays are particularly useful in the analysis of a multitype point pattern (a point pattern in which the points are identified as belonging to separate types). We may want to compute a summary function for the points of type i only, for each of the possible types i. This produces a $1 \times m$ array of functions. Alternatively we may compute a summary function for each possible pair of types (i, j). This produces an $m \times m$ array of functions.

For multitype point patterns the command `alltypes` will compute arrays of summary functions for each possible type or for each possible pair of types. The function `alltypes` returns an object of class `"fasp"`.

An object of class `"fasp"` is a list containing at least the following components:

fns A list of data frames, each representing one of the functions.

which A matrix representing the spatial arrangement of the functions. If `which[i,j] = k` then the function represented by `fns[[k]]` should be plotted in the panel at position (i, j). If `which[i,j] = NA` then nothing is plotted in that position.

titles A list of character strings, providing suitable plotting titles for the functions.

default.formulae A list of default formulae for plotting each of the functions.

title A character string, giving a default title for the array when it is plotted.

Functions available

There are methods for `plot`, `print` and `"["` for this class.

The plot method displays the entire array of functions. The method `[.fasp` selects a sub-array using the natural indices `i,j`.

The command `eval.fasp` can be used to apply a transformation to each function in the array, and to combine two arrays.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`alltypes`, `plot.fasp`, `[.fasp`, `eval.fasp`

Examples

```
# multitype point pattern
data(amacrine)
GG <- alltypes(amacrine, "G")
plot(GG)

# select the row corresponding to cells of type "on"
Gon <- GG["on", ]
plot(Gon)

# extract the G function for i = "on", j = "off"
Gonoff <- GG["on", "off", drop=TRUE]

# Fisher variance stabilising transformation
GGfish <- eval.fasp(asin(sqrt(GG)))
plot(GGfish)
```

Fest *Estimate the Empty Space Function or its Hazard Rate*

Description

Estimates the empty space function $F(r)$ or its hazard rate $h(r)$ from a point pattern in a window of arbitrary shape.

Usage

```
Fest(X, ..., eps, r=NULL, breaks=NULL,
     correction=c("rs", "km", "cs"),
     domain=NULL)

Fhazard(X, ...)
```

Arguments

`X`	The observed point pattern, from which an estimate of $F(r)$ will be computed. An object of class `ppp`, or data in any format acceptable to `as.ppp()`.
`...`	Extra arguments, passed from `Fhazard` to `Fest`. Extra arguments to `Fest` are ignored.
`eps`	Optional. A positive number. The resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
`r`	Optional. Numeric vector. The values of the argument r at which $F(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`breaks`	This argument is for internal use only.
`correction`	Optional. The edge correction(s) to be used to estimate $F(r)$. A vector of character strings selected from `"none"`, `"rs"`, `"km"`, `"cs"` and `"best"`. Alternatively `correction="all"` selects all options.
`domain`	Optional. Calculations will be restricted to this subset of the window. See Details.

Details

`Fest` computes an estimate of the empty space function $F(r)$, and `Fhazard` computes an estimate of its hazard rate $h(r)$.

The empty space function (also called the "*spherical contact distribution*" or the "*point-to-nearest-event*" distribution) of a stationary point process X is the cumulative distribution function F of the distance from a fixed point in space to the nearest point of X.

An estimate of F derived from a spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern (Cressie, 1991; Diggle, 1983; Ripley, 1988). In exploratory analyses, the estimate of F is a useful statistic summarising the sizes of gaps in the pattern. For inferential purposes, the estimate of F is usually compared to the true value of F for a completely random (Poisson) point process, which is

$$F(r) = 1 - e^{-\lambda \pi r^2}$$

where λ is the intensity (expected number of points per unit area). Deviations between the empirical and theoretical F curves may suggest spatial clustering or spatial regularity.

This algorithm estimates the empty space function F from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X`) may have arbitrary shape.

The argument `X` is interpreted as a point pattern object (of class `"ppp"`, see `ppp.object`) and can be supplied in any of the formats recognised by `as.ppp`.

The algorithm uses two discrete approximations which are controlled by the parameter `eps` and by the spacing of values of `r` respectively. (See below for details.) First-time users are strongly advised not to specify these arguments.

The estimation of F is hampered by edge effects arising from the unobservability of points of the random pattern outside the window. An edge correction is needed to reduce bias (Baddeley, 1998; Ripley, 1988). The edge corrections implemented here are the border method or "*reduced sample*" estimator, the spatial Kaplan-Meier estimator (Baddeley and Gill, 1997) and the Chiu-Stoyan estimator (Chiu and Stoyan, 1998).

Our implementation makes essential use of the distance transform algorithm of image processing (Borgefors, 1986). A fine grid of pixels is created in the observation window. The Euclidean distance between two pixels is approximated by the length of the shortest path joining them in the grid, where a path is a sequence of steps between adjacent pixels, and horizontal, vertical and diagonal steps have length 1, 1 and $\sqrt{2}$ respectively in pixel units. If the pixel grid is sufficiently fine then this is an accurate approximation.

The parameter `eps` is the pixel width of the rectangular raster used to compute the distance transform (see below). It must not be too large: the absolute error in distance values due to discretisation is bounded by `eps`.

If `eps` is not specified, the function checks whether the window `Window(X)` contains pixel raster information. If so, then `eps` is set equal to the pixel width of the raster; otherwise, `eps` defaults to 1/100 of the width of the observation window.

The argument `r` is the vector of values for the distance r at which $F(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of `hist`) for the computation of histograms of distances. The estimators are computed from histogram counts. This introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify `r`. However, if it is specified, `r` must satisfy `r[1] = 0`, and `max(r)` must be larger than the radius of the largest disc contained in the window. Furthermore, the spacing of successive `r` values must be very fine (ideally not greater than `eps/4`).

The algorithm also returns an estimate of the hazard rate function, $h(r)$ of $F(r)$. The hazard rate is defined by

$$h(r) = -\frac{d}{dr}\log(1 - F(r))$$

The hazard rate of F has been proposed as a useful exploratory statistic (Baddeley and Gill, 1994). The estimate of $h(r)$ given here is a discrete approximation to the hazard rate of the Kaplan-Meier estimator of F. Note that F is absolutely continuous (for any stationary point process X), so the hazard function always exists (Baddeley and Gill, 1997).

If the argument `domain` is given, the estimate of $F(r)$ will be based only on the empty space distances measured from locations inside `domain` (although their nearest data points may lie outside `domain`). This is useful in bootstrap techniques. The argument `domain` should be a window (object of class `"owin"`) or something acceptable to `as.owin`. It must be a subset of the window of the point pattern `X`.

The naive empirical distribution of distances from each location in the window to the nearest point of the data pattern, is a biased estimate of F. However this is also returned by the algorithm (if `correction="none"`), as it is sometimes useful in other contexts. Care should be taken not to use the uncorrected empirical F as if it were an unbiased estimator of F.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

The result of `Fest` is essentially a data frame containing up to seven columns:

`r`	the values of the argument r at which the function $F(r)$ has been estimated
`rs`	the "reduced sample" or "border correction" estimator of $F(r)$
`km`	the spatial Kaplan-Meier estimator of $F(r)$
`hazard`	the hazard rate $\lambda(r)$ of $F(r)$ by the spatial Kaplan-Meier method
`cs`	the Chiu-Stoyan estimator of $F(r)$

raw	the uncorrected estimate of $F(r)$, i.e. the empirical distribution of the distance from a random point in the window to the nearest point of the data pattern X
theo	the theoretical value of $F(r)$ for a stationary Poisson process of the same estimated intensity.

The result of Fhazard contains only three columns

r	the values of the argument r at which the hazard rate $h(r)$ has been estimated
hazard	the spatial Kaplan-Meier estimate of the hazard rate $h(r)$
theo	the theoretical value of $h(r)$ for a stationary Poisson process of the same estimated intensity.

Warnings

The reduced sample (border method) estimator of F is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r. Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of F is always nondecreasing but its maximum value may be less than 1.

The estimate of hazard rate $h(r)$ returned by the algorithm is an approximately unbiased estimate for the integral of $h()$ over the corresponding histogram cell. It may exhibit oscillations due to discretisation effects. We recommend modest smoothing, such as kernel smoothing with kernel width equal to the width of a histogram cell, using Smooth.fv.

Note

Sizeable amounts of memory may be needed during the calculation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37-78.

Baddeley, A.J. and Gill, R.D. The empty space hazard of a spatial pattern. Research Report 1994/3, Department of Mathematics, University of Western Australia, May 1994.

Baddeley, A.J. and Gill, R.D. Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25** (1997) 263-292.

Borgefors, G. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34** (1986) 344-371.

Chiu, S.N. and Stoyan, D. (1998) Estimators of distance distributions for spatial patterns. *Statistica Neerlandica* **52**, 239–246.

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

Gest, Jest, Kest, km.rs, reduced.sample, kaplan.meier

Examples

```
    Fc <- Fest(cells, 0.01)

    # Tip: don't use F for the left hand side!
    # That's an abbreviation for FALSE

    plot(Fc)

    # P-P style plot
    plot(Fc, cbind(km, theo) ~ theo)

    # The empirical F is above the Poisson F
    # indicating an inhibited pattern

    ## Not run:
    plot(Fc, . ~ theo)
    plot(Fc, asin(sqrt(.)) ~ asin(sqrt(theo)))

## End(Not run)
```

Fiksel *The Fiksel Interaction*

Description

Creates an instance of Fiksel's double exponential pairwise interaction point process model, which can then be fitted to point pattern data.

Usage

```
Fiksel(r, hc=NA, kappa)
```

Arguments

r	The interaction radius of the Fiksel model
hc	The hard core distance
kappa	The rate parameter

Details

Fiksel (1984) introduced a pairwise interaction point process with the following interaction function c. For two points u and v separated by a distance $d = ||u - v||$, the interaction $c(u, v)$ is equal to 0 if $d < h$, equal to 1 if $d > r$, and equal to

$$\exp(a \exp(-\kappa d))$$

if $h \leq d \leq r$, where h, r, κ, a are parameters.

A graph of this interaction function is shown in the Examples. The interpretation of the parameters is as follows.

- h is the hard core distance: distinct points are not permitted to come closer than a distance h apart.
- r is the interaction range: points further than this distance do not interact.
- κ is the rate or slope parameter, controlling the decay of the interaction as distance increases.
- a is the interaction strength parameter, controlling the strength and type of interaction. If a is zero, the process is Poisson. If a is positive, the process is clustered. If a is negative, the process is inhibited (regular).

The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Fiksel pairwise interaction is yielded by the function Fiksel(). See the examples below.

The parameters h, r and κ must be fixed and given in the call to Fiksel, while the canonical parameter a is estimated by ppm().

To estimate h, r andκ it is possible to use profilepl. The maximum likelihood estimator of$h$ is the minimum interpoint distance.

If the hard core distance argument hc is missing or NA, it will be estimated from the data when ppm is called. The estimated value of hc is the minimum nearest neighbour distance multiplied by $n/(n+1)$, where n is the number of data points.

See also Stoyan, Kendall and Mecke (1987) page 161.

Value

An object of class "interact" describing the interpoint interaction structure of the Fiksel process with interaction radius r, hard core distance hc and rate parameter kappa.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Fiksel, T. (1984) Estimation of parameterized pair potentials of marked and non-marked Gibbsian point processes. *Electronische Informationsverabeitung und Kybernetika* **20**, 270–278.

Stoyan, D, Kendall, W.S. and Mecke, J. (1987) *Stochastic geometry and its applications*. Wiley.

See Also

ppm, pairwise.family, ppm.object, StraussHard

Examples

```
Fiksel(r=1,hc=0.02, kappa=2)
# prints a sensible description of itself

data(spruces)
X <- unmark(spruces)

fit <- ppm(X ~ 1, Fiksel(r=3.5, kappa=1))
plot(fitin(fit))
```

Finhom	*Inhomogeneous Empty Space Function*

Description

Estimates the inhomogeneous empty space function of a non-stationary point pattern.

Usage

```
Finhom(X, lambda = NULL, lmin = NULL, ...,
      sigma = NULL, varcov = NULL,
      r = NULL, breaks = NULL, ratio = FALSE, update = TRUE)
```

Arguments

`X`	The observed data point pattern, from which an estimate of the inhomogeneous F function will be computed. An object of class "ppp" or in a format recognised by `as.ppp()`
`lambda`	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern `X`, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a `function(x,y)` which can be evaluated to give the intensity value at any location.
`lmin`	Optional. The minimum possible value of the intensity over the spatial domain. A positive numerical value.
`sigma,varcov`	Optional arguments passed to `density.ppp` to control the smoothing bandwidth, when `lambda` is estimated by kernel smoothing.
`...`	Extra arguments passed to `as.mask` to control the pixel resolution, or passed to `density.ppp` to control the smoothing bandwidth.
`r`	vector of values for the argument r at which the inhomogeneous K function should be evaluated. Not normally given by the user; there is a sensible default.
`breaks`	This argument is for internal use only.
`ratio`	Logical. If `TRUE`, the numerator and denominator of the estimate will also be saved, for use in analysing replicated point patterns.
`update`	Logical. If `lambda` is a fitted model (class "ppm" or "kppm") and `update=TRUE` (the default), the model will first be refitted to the data `X` (using `update.ppm` or `update.kppm`) before the fitted intensity is computed. If `update=FALSE`, the fitted intensity of the model will be computed without fitting it to `X`.

Details

This command computes estimates of the inhomogeneous F-function (van Lieshout, 2010) of a point pattern. It is the counterpart, for inhomogeneous spatial point patterns, of the empty space function F for homogeneous point patterns computed by `Fest`.

The argument `X` should be a point pattern (object of class "ppp").

The inhomogeneous F function is computed using the border correction, equation (6) in Van Lieshout (2010).

The argument `lambda` should supply the (estimated) values of the intensity function λ of the point process. It may be either

a numeric vector containing the values of the intensity function at the points of the pattern `X`.

a pixel image (object of class `"im"`) assumed to contain the values of the intensity function at all locations in the window.

a fitted point process model (object of class `"ppm"` or `"kppm"`) whose fitted *trend* can be used as the fitted intensity. (If `update=TRUE` the model will first be refitted to the data `X` before the trend is computed.)

a function which can be evaluated to give values of the intensity at any locations.

omitted: if `lambda` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

If `lambda` is a numeric vector, then its length should be equal to the number of points in the pattern `X`. The value `lambda[i]` is assumed to be the the (estimated) value of the intensity $\lambda(x_i)$ for the point x_i of the pattern X. Each value must be a positive number; `NA`'s are not allowed.

If `lambda` is a pixel image, the domain of the image should cover the entire window of the point pattern. If it does not (which may occur near the boundary because of discretisation error), then the missing pixel values will be obtained by applying a Gaussian blur to `lambda` using `blur`, then looking up the values of this blurred image for the missing locations. (A warning will be issued in this case.)

If `lambda` is a function, then it will be evaluated in the form `lambda(x,y)` where `x` and `y` are vectors of coordinates of the points of `X`. It should return a numeric vector with length equal to the number of points in `X`.

If `lambda` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate `lambda[i]` for the point `X[i]` is computed by removing `X[i]` from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point `X[i]`. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Author(s)

Original code by Marie-Colette van Lieshout. C implementation and R adaptation by Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Ege Rubak `<rubak@math.aau.dk>`.

References

Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.

Van Lieshout, M.N.M. (2010) A J-function for inhomogeneous point processes. *Statistica Neerlandica* **65**, 183–201.

See Also

`Ginhom`, `Jinhom`, `Fest`

Examples

```
  ## Not run:
  plot(Finhom(swedishpines, sigma=bw.diggle, adjust=2))

## End(Not run)
  plot(Finhom(swedishpines, sigma=10))
```

---

fitin.ppm *Extract the Interaction from a Fitted Point Process Model*

Description

Given a point process model that has been fitted to point pattern data, this function extracts the interpoint interaction part of the model as a separate object.

Usage

```
fitin(object)

## S3 method for class 'ppm'
fitin(object)

## S3 method for class 'profilepl'
fitin(object)
```

Arguments

`object`	A fitted point process model (object of class `"ppm"` or `"profilepl"`).

Details

An object of class `"ppm"` describes a fitted point process model. It contains information about the original data to which the model was fitted, the spatial trend that was fitted, the interpoint interaction that was fitted, and other data. See `ppm.object`) for details of this class.

The function `fitin` extracts from this model the information about the fitted interpoint interaction only. The information is organised as an object of class `"fii"` (fitted interpoint interaction). This object can be printed or plotted.

Users may find this a convenient way to plot the fitted interpoint interaction term, as shown in the Examples.

For a pairwise interaction, the plot of the fitted interaction shows the pair interaction function (the contribution to the probability density from a pair of points as a function of the distance between them). For a higher-order interaction, the plot shows the strongest interaction (the value most different from 1) that could ever arise at the given distance.

The fitted interaction coefficients can also be extracted from this object using `coef`.

Value

An object of class `"fii"` representing the fitted interpoint interaction. This object can be printed and plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

Methods for handling fitted interactions: methods.fii, reach.fii, as.interact.fii.

Background: ppm, ppm.object.

Examples

```
   # unmarked
   model <- ppm(swedishpines ~1, PairPiece(seq(3,19,by=4)))
   f <- fitin(model)
   f
   plot(f)

# extract fitted interaction coefficients
   coef(f)

   # multitype
   # fit the stationary multitype Strauss process to `amacrine'
   r <- 0.02 * matrix(c(1,2,2,1), nrow=2,ncol=2)
   model <- ppm(amacrine ~1, MultiStrauss(r))
   f <- fitin(model)
   f
   plot(f)
```

---

fitted.lppm — *Fitted Intensity for Point Process on Linear Network*

Description

Given a point process model fitted to a point pattern on a linear network, compute the fitted intensity of the model at the points of the pattern, or at the points of the quadrature scheme used to fit the model.

Usage

```
## S3 method for class 'lppm'
fitted(object, ...,
                        dataonly = FALSE, new.coef = NULL,
      leaveoneout = FALSE)
```

Arguments

`object`	Fitted point process model on a linear network (object of class "lppm").
`...`	Ignored.
`dataonly`	Logical value indicating whether to computed fitted intensities at the points of the original point pattern dataset (dataonly=TRUE) or at all the quadrature points of the quadrature scheme used to fit the model (dataonly=FALSE, the default).

new.coef Numeric vector of parameter values to replace the fitted model parameters coef(object).

leaveoneout Logical. If TRUE the fitted value at each data point will be computed using a leave-one-out method. See Details.

Details

This is a method for the generic function fitted for the class "lppm" of fitted point process models on a linear network.

The locations u at which the fitted conditional intensity/trend is evaluated, are the points of the quadrature scheme used to fit the model in ppm. They include the data points (the points of the original point pattern dataset x) and other "dummy" points in the window of observation.

If leaveoneout=TRUE, fitted values will be computed for the data points only, using a 'leave-one-out' rule: the fitted value at X[i] is effectively computed by deleting this point from the data and re-fitting the model to the reduced pattern X[-i], then predicting the value at X[i]. (Instead of literally performing this calculation, we apply a Taylor approximation using the influence function computed in dfbetas.ppm.

Value

A vector containing the values of the fitted spatial trend.

Entries in this vector correspond to the quadrature points (data or dummy points) used to fit the model. The quadrature points can be extracted from object by union.quad(quad.ppm(object)).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

lppm, predict.lppm

Examples

```
fit <- lppm(spiders~x+y)
a <- fitted(fit)
b <- fitted(fit, dataonly=TRUE)
```

---

fitted.mppm *Fitted Conditional Intensity for Multiple Point Process Model*

Description

Given a point process model fitted to multiple point patterns, compute the fitted conditional intensity of the model at the points of each data pattern, or at the points of the quadrature schemes used to fit the model.

Usage

```
## S3 method for class 'mppm'
fitted(object, ..., type = "lambda", dataonly = FALSE)
```

Arguments

object	The fitted model. An object of class "mppm" obtained from mppm.
...	Ignored.
type	Type of fitted values: either "trend" for the spatial trend, or "lambda" or "cif" for the conditional intensity.
dataonly	If TRUE, fitted values are computed only for the points of the data point patterns. If FALSE, fitted values are computed for the points of the quadrature schemes used to fit the model.

Details

This function evaluates the conditional intensity $\hat{\lambda}(u,x)$ or spatial trend $b(\hat{u})$ of the fitted point process model for certain locations u, for each of the original point patterns x to which the model was fitted.

The locations u at which the fitted conditional intensity/trend is evaluated, are the points of the quadrature schemes used to fit the model in mppm. They include the data points (the points of the original point pattern datasets) and other "dummy" points in the window of observation.

Use predict.mppm to compute the fitted conditional intensity at other locations or with other values of the explanatory variables.

Value

A list of vectors (one for each row of the original hyperframe, i.e. one vector for each of the original point patterns) containing the values of the fitted conditional intensity or (if type="trend") the fitted spatial trend.

Entries in these vector correspond to the quadrature points (data or dummy points) used to fit the model. The quadrature points can be extracted from object by quad.mppm(object).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

See Also

mppm, predict.mppm

Examples

```
model <- mppm(Bugs ~ x, data=hyperframe(Bugs=waterstriders),
                        interaction=Strauss(7))
cifs <- fitted(model)
```

`fitted.ppm` *Fitted Conditional Intensity for Point Process Model*

Description

Given a point process model fitted to a point pattern, compute the fitted conditional intensity or fitted trend of the model at the points of the pattern, or at the points of the quadrature scheme used to fit the model.

Usage

```
## S3 method for class 'ppm'
fitted(object, ..., type="lambda", dataonly=FALSE,
  new.coef=NULL, leaveoneout=FALSE, drop=FALSE, check=TRUE, repair=TRUE,
  dropcoef=FALSE)
```

Arguments

`object`	The fitted point process model (an object of class "ppm")
`...`	Ignored.
`type`	String (partially matched) indicating whether the fitted value is the conditional intensity ("lambda" or "cif") or the first order trend ("trend") or the logarithm of conditional intensity ("link").
`dataonly`	Logical. If TRUE, then values will only be computed at the points of the data point pattern. If FALSE, then values will be computed at all the points of the quadrature scheme used to fit the model, including the points of the data point pattern.
`new.coef`	Numeric vector of parameter values to replace the fitted model parameters coef(object).
`leaveoneout`	Logical. If TRUE the fitted value at each data point will be computed using a leave-one-out method. See Details.
`drop`	Logical value determining whether to delete quadrature points that were not used to fit the model.
`check`	Logical value indicating whether to check the internal format of object. If there is any possibility that this object has been restored from a dump file, or has otherwise lost track of the environment where it was originally computed, set check=TRUE.
`repair`	Logical value indicating whether to repair the internal format of object, if it is found to be damaged.
`dropcoef`	Internal use only.

Details

The argument `object` must be a fitted point process model (object of class `"ppm"`). Such objects are produced by the model-fitting algorithm `ppm`).

This function evaluates the conditional intensity $\hat{\lambda}(u, x)$ or spatial trend $\hat{b}(u)$ of the fitted point process model for certain locations u, where `x` is the original point pattern dataset to which the model was fitted.

The locations u at which the fitted conditional intensity/trend is evaluated, are the points of the quadrature scheme used to fit the model in `ppm`. They include the data points (the points of the original point pattern dataset `x`) and other "dummy" points in the window of observation.

If `leaveoneout=TRUE`, fitted values will be computed for the data points only, using a 'leave-one-out' rule: the fitted value at `X[i]` is effectively computed by deleting this point from the data and re-fitting the model to the reduced pattern `X[-i]`, then predicting the value at `X[i]`. (Instead of literally performing this calculation, we apply a Taylor approximation using the influence function computed in `dfbetas.ppm`.

The argument `drop` is explained in `quad.ppm`.

Use `predict.ppm` to compute the fitted conditional intensity at other locations or with other values of the explanatory variables.

Value

A vector containing the values of the fitted conditional intensity, fitted spatial trend, or logarithm of the fitted conditional intensity.

Entries in this vector correspond to the quadrature points (data or dummy points) used to fit the model. The quadrature points can be extracted from `object` by `union.quad(quad.ppm(object))`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005). Residual analysis for spatial point processes (with discussion). *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

See Also

`ppm.object`, `ppm`, `predict.ppm`

Examples

```
str <- ppm(cells ~x, Strauss(r=0.1))
lambda <- fitted(str)

# extract quadrature points in corresponding order
quadpoints <- union.quad(quad.ppm(str))

# plot conditional intensity values
# as circles centred on the quadrature points
quadmarked <- setmarks(quadpoints, lambda)
plot(quadmarked)
```

```
if(!interactive()) str <- ppm(cells ~ x)

lambdaX <- fitted(str, leaveoneout=TRUE)
```

`fitted.slrm` *Fitted Probabilities for Spatial Logistic Regression*

Description

Given a fitted Spatial Logistic Regression model, this function computes the fitted probabilities for each pixel.

Usage

```
## S3 method for class 'slrm'
fitted(object, ...)
```

Arguments

`object`	a fitted spatial logistic regression model. An object of class `"slrm"`.
`...`	Ignored.

Details

This is a method for the generic function `fitted` for spatial logistic regression models (objects of class `"slrm"`, usually obtained from the function `slrm`).

The algorithm computes the fitted probabilities of the presence of a random point in each pixel.

Value

A pixel image (object of class `"im"`) containing the fitted probability for each pixel.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`slrm`, `fitted`

Examples

```
X <- rpoispp(42)
fit <- slrm(X ~ x+y)
plot(fitted(fit))
```

fixef.mppm *Extract Fixed Effects from Point Process Model*

Description

Given a point process model fitted to a list of point patterns, extract the fixed effects of the model. A method for `fixef`.

Usage

```
## S3 method for class 'mppm'
fixef(object, ...)
```

Arguments

`object`	A fitted point process model (an object of class "mppm").
`...`	Ignored.

Details

This is a method for the generic function `fixef`.

The argument `object` must be a fitted point process model (object of class "mppm") produced by the fitting algorithm `mppm`). This represents a point process model that has been fitted to a list of several point pattern datasets. See `mppm` for information.

This function extracts the coefficients of the fixed effects of the model.

Value

A numeric vector of coefficients.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented in **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

See Also

`coef.mppm`

Examples

```
H <- hyperframe(Y = waterstriders)
# Tweak data to exaggerate differences
H$Y[[1]] <- rthin(H$Y[[1]], 0.3)
m1 <- mppm(Y ~ id,  data=H, Strauss(7))
fixef(m1)
m2 <- mppm(Y ~ 1,  random=~1|id, data=H, Strauss(7))
fixef(m2)
```

`flipxy` *Exchange X and Y Coordinates*

Description

Exchanges the x and y coordinates in a spatial dataset.

Usage

```
 flipxy(X)
 ## S3 method for class 'owin'
flipxy(X)
 ## S3 method for class 'ppp'
flipxy(X)
 ## S3 method for class 'psp'
flipxy(X)
 ## S3 method for class 'im'
flipxy(X)
```

Arguments

`X`	Spatial dataset. An object of class `"owin"`, `"ppp"`, `"psp"` or `"im"`.

Details

This function swaps the x and y coordinates of a spatial dataset. This could also be performed using the command `affine`, but `flipxy` is faster.

The function `flipxy` is generic, with methods for the classes of objects listed above.

Value

Another object of the same type, representing the result of swapping the x and y coordinates.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`affine`, `reflect`, `rotate`, `shift`

Examples

```
data(cells)
X <- flipxy(cells)
```

FmultiInhom *Inhomogeneous Marked F-Function*

Description

For a marked point pattern, estimate the inhomogeneous version of the multitype F function, effectively the cumulative distribution function of the distance from a fixed point to the nearest point in subset J, adjusted for spatially varying intensity.

Usage

```
FmultiInhom(X, J,
            lambda = NULL, lambdaJ = NULL, lambdamin = NULL,
            ...,
            r = NULL)
```

Arguments

X	A spatial point pattern (object of class "ppp".
J	A subset index specifying the subset of points to which distances are measured. Any kind of subset index acceptable to [.ppp.
lambda	Intensity estimates for each point of X. A numeric vector of length equal to npoints(X). Incompatible with lambdaJ.
lambdaJ	Intensity estimates for each point of X[J]. A numeric vector of length equal to npoints(X[J]). Incompatible with lambda.
lambdamin	A lower bound for the intensity, or at least a lower bound for the values in lambdaJ or lambda[J].
...	Ignored.
r	Vector of distance values at which the inhomogeneous G function should be estimated. There is a sensible default.

Details

See Cronie and Van Lieshout (2015).

Value

Object of class "fv" containing the estimate of the inhomogeneous multitype F function.

Author(s)

Ottmar Cronie and Marie-Colette van Lieshout. Rewritten for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.

References

Cronie, O. and Van Lieshout, M.N.M. (2015) Summary statistics for inhomogeneous marked point processes. *Annals of the Institute of Statistical Mathematics* DOI: 10.1007/s10463-015-0515-z

See Also

Finhom

Examples

```
X <- amacrine
J <- (marks(X) == "off")
mod <- ppm(X ~ marks * x)
lam <- fitted(mod, dataonly=TRUE)
lmin <- min(predict(mod)[["off"]]) * 0.9
plot(FmultiInhom(X, J, lambda=lam, lambdamin=lmin))
```

---

foo *Foo is Not a Real Name*

Description

The name `foo` is not a real name: it is a place holder, used to represent the name of any desired thing.

The functions defined here simply print an explanation of the placeholder name `foo`.

Usage

```
foo()

## S3 method for class 'foo'
plot(x, ...)
```

Arguments

`x`	Ignored.
`...`	Ignored.

Details

The name `foo` is used by computer scientists as a *place holder*, to represent the name of any desired object or function. It is not the name of an actual object or function; it serves only as an example, to explain a concept.

However, many users misinterpret this convention, and actually type the command `foo` or `foo()`. Then they email the package author to inform them that `foo` is not defined.

To avoid this correspondence, we have now defined an object called `foo`.

The function `foo()` prints a message explaining that `foo` is not really the name of a variable.

The function can be executed simply by typing `foo` without parentheses.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

beginner

Examples

```
foo
```

`formula.fv` *Extract or Change the Plot Formula for a Function Value Table*

Description

Extract or change the default plotting formula for an object of class "fv" (function value table).

Usage

```
## S3 method for class 'fv'
formula(x, ...)

formula(x, ...) <- value

## S3 replacement method for class 'fv'
formula(x, ...) <- value
```

Arguments

x	An object of class "fv", containing the values of several estimates of a function.
...	Arguments passed to other methods.
value	New value of the formula. Either a formula or a character string.

Details

A function value table (object of class "fv", see fv.object) is a convenient way of storing and plotting several different estimates of the same function.

The default behaviour of plot(x) for a function value table x is determined by a formula associated with x called its *plot formula*. See plot.fv for explanation about these formulae.

The function formula.fv is a method for the generic command formula. It extracts the plot formula associated with the object.

The function formula<- is generic. It changes the formula associated with an object.

The function formula<-.fv is the method for formula<- for the class "fv". It changes the plot formula associated with the object.

Value

The result of `formula.fv` is a character string containing the plot formula. The result of `formula<-.fv` is a new object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

fv, plot.fv, formula.

Examples

```
K <- Kest(cells)
formula(K)
formula(K) <- (iso ~ r)
```

---

formula.ppm *Model Formulae for Gibbs Point Process Models*

Description

Extract the trend formula, or the terms in the trend formula, in a fitted Gibbs point process model.

Usage

```
## S3 method for class 'ppm'
formula(x, ...)
## S3 method for class 'ppm'
terms(x, ...)
```

Arguments

x	An object of class "ppm", representing a fitted point process model.
...	Arguments passed to other methods.

Details

These functions are methods for the generic commands formula and terms for the class "ppm".

An object of class "ppm" represents a fitted Poisson or Gibbs point process model. It is obtained from the model-fitting function ppm.

The method `formula.ppm` extracts the trend formula from the fitted model `x` (the formula originally specified as the argument `trend` to ppm). The method `terms.ppm` extracts the individual terms in the trend formula.

Value

See the help files for the corresponding generic functions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

ppm, as.owin, coef.ppm, extractAIC.ppm, fitted.ppm, logLik.ppm, model.frame.ppm, model.matrix.ppm, plot.ppm, predict.ppm, residuals.ppm, simulate.ppm, summary.ppm, update.ppm, vcov.ppm.

Examples

```
data(cells)
fit <- ppm(cells, ~x)
formula(fit)
terms(fit)
```

---

fourierbasis *Fourier Basis Functions*

Description

Evaluates the Fourier basis functions on a d-dimensional box with d-dimensional frequencies k_i at the d-dimensional coordinates x_j.

Usage

```
fourierbasis(x, k, win = boxx(rep(list(0:1), ncol(k))))
fourierbasisraw(x, k, boxlengths)
```

Arguments

x	Coordinates. A data.frame or matrix with n rows and d columns giving the d-dimensional coordinates.
k	Frequencies. A data.frame or matrix with m rows and d columns giving the frequencies of the Fourier-functions.
win	window (of class "owin", "box3" or "boxx") giving the d-dimensional box domain of the Fourier functions.
boxlengths	numeric giving the side lengths of the box domain of the Fourier functions.

Details

The result is an m by n matrix where the (i, j)'th entry is the d-dimensional Fourier basis function with frequency k_i evaluated at the point x_j, i.e.,

$$\frac{1}{\sqrt{|W|}} \exp(2\pi i \sum l = 1^d k_{i,l} x_{j,l} / L_l)$$

where $L_l, l = 1, ..., d$ are the box side lengths and $|W|$ is the volume of the domain (window/box). Note that the algorithm does not check whether the coordinates given in x are contained in the given box. Actually the box is only used to determine the side lengths and volume of the domain for normalization.

The stripped down faster version fourierbasisraw doesn't do checking or conversion of arguments and requires x and k to be matrices.

Value

An m by n matrix of complex values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
## 27 rows of three dimensional Fourier frequencies:
k <- expand.grid(-1:1,-1:1, -1:1)
## Two random points in the three dimensional unit box:
x <- rbind(runif(3),runif(3))
## 27 by 2 resulting matrix:
v <- fourierbasis(x, k)
head(v)
```

Frame *Extract or Change the Containing Rectangle of a Spatial Object*

Description

Given a spatial object (such as a point pattern or pixel image) in two dimensions, these functions extract or change the containing rectangle inside which the object is defined.

Usage

```
   Frame(X)

   ## Default S3 method:
Frame(X)

   Frame(X) <- value

   ## S3 replacement method for class 'owin'
Frame(X) <- value

   ## S3 replacement method for class 'ppp'
Frame(X) <- value

   ## S3 replacement method for class 'im'
Frame(X) <- value

   ## Default S3 replacement method:
Frame(X) <- value
```

Arguments

X	A spatial object such as a point pattern, line segment pattern or pixel image.
value	A rectangular window (object of class "owin" of type "rectangle") to be used as the new containing rectangle for X.

Details

The functions Frame and Frame<- are generic.

Frame(X) extracts the rectangle inside which X is defined.

Frame(X) <- R changes the rectangle inside which X is defined to the new rectangle R.

Value

The result of Frame is a rectangular window (object of class "owin" of type "rectangle").

The result of Frame<- is the updated object X, of the same class as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

Window

Examples

```
Frame(cells)
X <- demopat
Frame(X)
Frame(X) <- owin(c(0, 11000), c(400, 8000))
```

---

fryplot *Fry Plot of Point Pattern*

Description

Displays the Fry plot (Patterson plot) of a spatial point pattern.

Usage

```
fryplot(X, ..., width=NULL, from=NULL, to=NULL, axes=FALSE)
frypoints(X, from=NULL, to=NULL, dmax=Inf)
```

Arguments

`X`	A point pattern (object of class "ppp") or something acceptable to `as.ppp`.
`...`	Optional arguments to control the appearance of the plot.
`width`	Optional parameter indicating the width of a box for a zoomed-in view of the Fry plot near the origin.
`from,to`	Optional. Subset indices specifying which points of `X` will be considered when forming the vectors (drawn from each point of `from`, to each point of `to`.)
`axes`	Logical value indicating whether to draw axes, crossing at the origin.
`dmax`	Maximum distance between points. Pairs at greater distances do not contribute to the result. The default means there is no maximum distance.

Details

The function `fryplot` generates a Fry plot (or Patterson plot); `frypoints` returns the points of the Fry plot as a point pattern dataset.

Fry (1979) and Hanna and Fry (1979) introduced a manual graphical method for investigating features of a spatial point pattern of mineral deposits. A transparent sheet, marked with an origin or centre point, is placed over the point pattern. The transparent sheet is shifted so that the origin lies over one of the data points, and the positions of all the *other* data points are copied onto the transparent sheet. This procedure is repeated for each data point in turn. The resulting plot (the Fry plot) is a pattern of $n(n-1)$ points, where n is the original number of data points. This procedure was previously proposed by Patterson (1934, 1935) for studying inter-atomic distances in crystals, and is also known as a Patterson plot.

The function `fryplot` generates the Fry/Patterson plot. Standard graphical parameters such as `main`, `pch`, `lwd`, `col`, `bg`, `cex` can be used to control the appearance of the plot. To zoom in (to view only a subset of the Fry plot at higher magnification), use the argument `width` to specify the width of a rectangular field of view centred at the origin, or the standard graphical arguments `xlim` and `ylim` to specify another rectangular field of view. (The actual field of view may be slightly larger, depending on the graphics device.)

The function `frypoints` returns the points of the Fry plot as a point pattern object. There may be a large number of points in this pattern, so this function should be used only if further analysis of the Fry plot is required.

Fry plots are particularly useful for recognising anisotropy in regular point patterns. A void around the origin in the Fry plot suggests regularity (inhibition between points) and the shape of the void gives a clue to anisotropy in the pattern. Fry plots are also useful for detecting periodicity or rounding of the spatial coordinates.

In mathematical terms, the Fry plot of a point pattern `X` is simply a plot of the vectors `X[i] - X[j]` connecting all pairs of distinct points in `X`.

The Fry plot is related to the K function (see `Kest`) and the reduced second moment measure (see `Kmeasure`). For example, the number of points in the Fry plot lying within a circle of given radius is an unnormalised and uncorrected version of the K function. The Fry plot has a similar appearance to the plot of the reduced second moment measure `Kmeasure` when the smoothing parameter `sigma` is very small.

The Fry plot does not adjust for the effect of the size and shape of the sampling window. The density of points in the Fry plot tapers off near the edges of the plot. This is an edge effect, a consequence of the bounded sampling window. In geological applications this is usually not important, because interest is focused on the behaviour near the origin where edge effects can be ignored. To correct for the edge effect, use `Kmeasure` or `Kest` or its relatives.

Value

fryplot returns NULL. frypoints returns a point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Fry, N. (1979) Random point distributions and strain measurement in rocks. *Tectonophysics* **60**, 89–105.

Hanna, S.S. and Fry, N. (1979) A comparison of methods of strain determination in rocks from southwest Dyfed (Pembrokeshire) and adjacent areas. *Journal of Structural Geology* **1**, 155–162.

Patterson, A.L. (1934) A Fourier series method for the determination of the component of inter-atomic distances in crystals. *Physics Reviews* **46**, 372–376.

Patterson, A.L. (1935) A direct method for the determination of the components of inter-atomic distances in crystals. *Zeitschrift fuer Krystallographie* **90**, 517–554.

See Also

Kmeasure, Kest

Examples

```
## unmarked data
fryplot(cells)
Y <- frypoints(cells)

## numerical marks
fryplot(longleaf, width=4, axes=TRUE)

## multitype points
fryplot(amacrine, width=0.2,
                  from=(marks(amacrine) == "on"),
                  chars=c(3,16), cols=2:3,
                  main="Fry plot centred at an On-cell")
points(0,0)
```

---

funxy *Spatial Function Class*

Description

A simple class of functions of spatial location

Usage

```
funxy(f, W)
```

Arguments

- `f` — A `function` in the R language with arguments `x,y` (at least)
- `W` — Window (object of class `"owin"`) inside which the function is well-defined.

Details

This creates an object of class `"funxy"`. This is a simple mechanism for handling a function of spatial location $f(x, y)$ to make it easier to display and manipulate.

`f` should be a `function` in the R language. The first two arguments of `f` must be named `x` and `y` respectively.

`W` should be a window (object of class `"owin"`) inside which the function `f` is well-defined.

The function `f` should be vectorised: that is, if `x` and `y` are numeric vectors of the same length `n`, then `v <- f(x,y)` should be a vector of length `n`.

The resulting function `g <- funxy(f, W)` has the same formal arguments as `f`. It accepts numeric vectors `x,y` as described above, but if `y` is missing, then `x` may be a point pattern (object of class `"ppp"` or `"lpp"`) from which the coordinates should be extracted.

Value

A `function` with the same arguments as `f`, which also belongs to the class `"funxy"`. This class has methods for `print`, `plot`, `contour` and `persp`.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`r.turner@auckland.ac.nz`> and Ege Rubak <`rubak@math.aau.dk`>.

See Also

`plot.funxy`

Examples

```
f <- function(x,y) { x^2 + y^2 - 1}
g <- funxy(f, square(2))
g(0.2, 0.3)
g
g(cells[1:4])
```

---

fv — *Create a Function Value Table*

Description

Advanced Use Only. This low-level function creates an object of class `"fv"` from raw numerical data.

Usage

```
fv(x, argu = "r", ylab = NULL, valu, fmla = NULL, alim = NULL,
   labl = names(x), desc = NULL, unitname = NULL, fname = NULL, yexp = ylab)
```

Arguments

`x`	A data frame with at least 2 columns containing the values of the function argument and the corresponding values of (one or more versions of) the function.
`argu`	String. The name of the column of `x` that contains the values of the function argument.
`ylab`	Either `NULL`, or an R language expression representing the mathematical name of the function. See Details.
`valu`	String. The name of the column of `x` that should be taken as containing the function values, in cases where a single column is required.
`fmla`	Either `NULL`, or a `formula` specifying the default plotting behaviour. See Details.
`alim`	Optional. The default range of values of the function argument for which the function will be plotted. Numeric vector of length 2.
`labl`	Optional. Plot labels for the columns of `x`. A vector of strings, with one entry for each column of `x`.
`desc`	Optional. Descriptions of the columns of `x`. A vector of strings, with one entry for each column of `x`.
`unitname`	Optional. Name of the unit (usually a unit of length) in which the function argument is expressed. Either a single character string, or a vector of two character strings giving the singular and plural forms, respectively.
`fname`	Optional. The name of the function itself. A character string.
`yexp`	Optional. Alternative form of `ylab` more suitable for annotating an axis of the plot. See Details.

Details

This documentation is provided for experienced programmers who want to modify the internal behaviour of **spatstat**. Other users please see `fv.object`.

The low-level function `fv` is used to create an object of class `"fv"` from raw numerical data.

The data frame `x` contains the numerical data. It should have one column (typically but not necessarily named `"r"`) giving the values of the function argument for which the function has been evaluated; and at least one other column, containing the corresponding values of the function.

Typically there is more than one column of function values. These columns typically give the values of different versions or estimates of the same function, for example, different estimates of the K function obtained using different edge corrections. However they may also contain the values of related functions such as the derivative or hazard rate.

`argu` specifies the name of the column of `x` that contains the values of the function argument (typically `argu="r"` but this is not compulsory).

`valu` specifies the name of another column that contains the 'recommended' estimate of the function. It will be used to provide function values in those situations where a single column of data is required. For example, `envelope` computes its simulation envelopes using the recommended value of the summary function.

`fmla` specifies the default plotting behaviour. It should be a formula, or a string that can be converted to a formula. Variables in the formula are names of columns of `x`. See `plot.fv` for the interpretation of this formula.

`alim` specifies the recommended range of the function argument. This is used in situations where statistical theory or statistical practice indicates that the computed estimates of the function are not trustworthy outside a certain range of values of the function argument. By default, plot.fv will restrict the plot to this range.

fname is a string giving the name of the function itself. For example, the K function would have fname="K".

ylab is a mathematical expression for the function value, used when labelling an axis of the plot, or when printing a description of the function. It should be an R language object. For example the K function's mathematical name $K(r)$ is rendered by ylab=quote(K(r)).

If yexp is present, then ylab will be used only for printing, and yexp will be used for annotating axes in a plot. (Otherwise yexp defaults to ylab). For example the cross-type K function $K_{1,2}(r)$ is rendered by something like ylab=quote(Kcross[1,2](r)) and yexp=quote(Kcross[list(1,2)](r)) to get the most satisfactory behaviour.

(A useful tip: use substitute instead of quote to insert values of variables into an expression, e.g. substitute(Kcross[i,j](r), list(i=42,j=97)) yields the same as quote(Kcross[42, 97](r)).)

labl is a character vector specifying plot labels for each column of x. These labels will appear on the plot axes (in non-default plots), legends and printed output. Entries in labl may contain the string "%s" which will be replaced by fname. For example the border-corrected estimate of the K function has label "%s[bord](r)" which becomes "K[bord](r)".

desc is a character vector containing intelligible explanations of each column of x. Entries in desc may contain the string "%s" which will be replaced by ylab. For example the border correction estimate of the K function has description "border correction estimate of %s".

Value

An object of class "fv", see fv.object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

See plot.fv for plotting an "fv" object.

See as.function.fv to convert an "fv" object to an R function.

Use cbind.fv to combine several "fv" objects. Use bind.fv to glue additional columns onto an existing "fv" object.

Use range.fv to compute the range of y values for a function, and with.fv for more complicated calculations.

The functions fvnames, fvnames<- allow the user to use standard abbreviations to refer to columns of an "fv" object.

Undocumented functions for modifying an "fv" object include tweak.fv.entry and rebadge.fv.

Examples

```
df <- data.frame(r=seq(0,5,by=0.1))
df <- transform(df, a=pi*r^2, b=3*r^2)
X <- fv(df, "r", quote(A(r)),
           "a", cbind(a, b) ~ r,
           alim=c(0,4),
           labl=c("r", "%s[true](r)", "%s[approx](r)"),
```

```
                desc=c("radius of circle",
                       "true area %s",
                       "rough area %s"),
                fname="A")
   X
```

fv.object *Function Value Table*

Description

A class "fv" to support the convenient plotting of several estimates of the same function.

Details

An object of this class is a convenient way of storing and plotting several different estimates of the same function.

It is a data frame with extra attributes indicating the recommended way of plotting the function, and other information.

There are methods for print and plot for this class.

Objects of class "fv" are returned by Fest, Gest,Jest, and Kest along with many other functions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Objects of class "fv" are returned by Fest, Gest,Jest, and Kest along with many other functions.

See plot.fv for plotting an "fv" object.

See as.function.fv to convert an "fv" object to an R function.

Use cbind.fv to combine several "fv" objects. Use bind.fv to glue additional columns onto an existing "fv" object.

Undocumented functions for modifying an "fv" object include fvnames, fvnames<-, tweak.fv.entry and rebadge.fv.

Examples

```
data(cells)
K <- Kest(cells)

class(K)

K  # prints a sensible summary

plot(K)
```

`fvnames` *Abbreviations for Groups of Columns in Function Value Table*

Description

Groups of columns in a function value table (object of class "fv") identified by standard abbreviations.

Usage

```
fvnames(X, a = ".")

fvnames(X, a = ".") <- value
```

Arguments

X	Function value table (object of class "fv"). See fv.object.
a	One of the standard abbreviations listed below.
value	Character vector containing names of columns of X.

Details

An object of class "fv" represents a table of values of a function, usually a summary function for spatial data such as the K-function, for which several different statistical estimators may be available. The different estimates are stored as columns of the table.

Auxiliary information carried in the object X specifies some columns or groups of columns of this table that should be used for particular purposes. For convenience these groups can be referred to by standard abbreviations which are recognised by various functions in the **spatstat** package, such as plot.fv.

These abbreviations are:

".x"	the function argument
".y"	the recommended value of the function
"."	all function values to be plotted by default (in order of plotting)
".s"	the upper and lower limits of shading (for envelopes and confidence intervals)
".a"	all function values

The command fvnames(X, a) expands the abbreviation a and returns a character vector containing the names of the columns.

The assignment fvnames(X, a) <- value changes the definition of the abbreviation a to the character vector value. It does not change the labels of any columns.

Note that fvnames(x, ".") lists the columns of values that will be plotted by default, in the order that they would be plotted, not in order of the column position. The order in which curves are plotted affects the colours and line styles associated with the curves.

Value

For `fvnames`, a character vector.

For `fvnames<-`, the updated object.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`fv.object`, `plot.fv`

Examples

```
K <- Kest(cells)
fvnames(K, ".y")
fvnames(K, ".y") <- "trans"
```

`G3est` *Nearest Neighbour Distance Distribution Function of a Three-Dimensional Point Pattern*

Description

Estimates the nearest-neighbour distance distribution function $G_3(r)$ from a three-dimensional point pattern.

Usage

```
G3est(X, ..., rmax = NULL, nrval = 128, correction = c("rs", "km", "Hanisch"))
```

Arguments

`X`	Three-dimensional point pattern (object of class `"pp3"`).
`...`	Ignored.
`rmax`	Optional. Maximum value of argument r for which $G_3(r)$ will be estimated.
`nrval`	Optional. Number of values of r for which $G_3(r)$ will be estimated. A large value of `nrval` is required to avoid discretisation effects.
`correction`	Optional. Character vector specifying the edge correction(s) to be applied. See Details.

Details

For a stationary point process Φ in three-dimensional space, the nearest-neighbour function is

$$G_3(r) = P(d^*(x, \Phi) \leq r \mid x \in \Phi)$$

the cumulative distribution function of the distance $d^*(x, \Phi)$ from a typical point x in Φ to its nearest neighbour, i.e. to the nearest *other* point of Φ.

The three-dimensional point pattern `X` is assumed to be a partial realisation of a stationary point process Φ. The nearest neighbour function of Φ can then be estimated using techniques described in the References. For each data point, the distance to the nearest neighbour is computed. The empirical cumulative distribution function of these values, with appropriate edge corrections, is the estimate of $G_3(r)$.

The available edge corrections are:

`"rs"`: the reduced sample (aka minus sampling, border correction) estimator (Baddeley et al, 1993)

`"km"`: the three-dimensional version of the Kaplan-Meier estimator (Baddeley and Gill, 1997)

`"Hanisch"`: the three-dimensional generalisation of the Hanisch estimator (Hanisch, 1984).

Alternatively `correction="all"` selects all options.

Value

A function value table (object of class `"fv"`) that can be plotted, printed or coerced to a data frame containing the function values.

Warnings

A large value of `nrval` is required in order to avoid discretisation effects (due to the use of histograms in the calculation).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rana Moyeed.

References

Baddeley, A.J, Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.

Baddeley, A.J. and Gill, R.D. (1997) Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25**, 263–292.

Hanisch, K.-H. (1984) Some remarks on estimators of the distribution function of nearest neighbour distance in stationary spatial point patterns. *Mathematische Operationsforschung und Statistik, series Statistics* **15**, 409–412.

See Also

`F3est`, `K3est`, `pcf3est`

Examples

```
X <- rpoispp3(42)
Z <- G3est(X)
if(interactive()) plot(Z)
```

`gauss.hermite` *Gauss-Hermite Quadrature Approximation to Expectation for Normal Distribution*

Description

Calculates an approximation to the expected value of any function of a normally-distributed random variable, using Gauss-Hermite quadrature.

Usage

```
gauss.hermite(f, mu = 0, sd = 1, ..., order = 5)
```

Arguments

`f`	The function whose moment should be approximated.
`mu`	Mean of the normal distribution.
`sd`	Standard deviation of the normal distribution.
`...`	Additional arguments passed to `f`.
`order`	Number of quadrature points in the Gauss-Hermite quadrature approximation. A small positive integer.

Details

This algorithm calculates the approximate expected value of `f(Z)` when `Z` is a normally-distributed random variable with mean `mu` and standard deviation `sd`. The expected value is an integral with respect to the Gaussian density; this integral is approximated using Gauss-Hermite quadrature.

The argument `f` should be a function in the R language whose first argument is the variable `Z`. Additional arguments may be passed through `...`. The value returned by `f` may be a single numeric value, a vector, or a matrix. The values returned by `f` for different values of `Z` must have compatible dimensions.

The result is a weighted average of several values of `f`.

Value

Numeric value, vector or matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

Examples

```
gauss.hermite(function(x) x^2, 3, 1)
```

Gcom *Model Compensator of Nearest Neighbour Function*

Description

Given a point process model fitted to a point pattern dataset, this function computes the *compensator* of the nearest neighbour distance distribution function G based on the fitted model (as well as the usual nonparametric estimates of G based on the data alone). Comparison between the nonparametric and model-compensated G functions serves as a diagnostic for the model.

Usage

```
Gcom(object, r = NULL, breaks = NULL, ...,
     correction = c("border", "Hanisch"),
     conditional = !is.poisson(object),
     restrict=FALSE,
     model=NULL,
     trend = ~1, interaction = Poisson(),
     rbord = reach(interaction),
     ppmcorrection="border",
     truecoef = NULL, hi.res = NULL)
```

Arguments

- `object` — Object to be analysed. Either a fitted point process model (object of class "ppm") or a point pattern (object of class "ppp") or quadrature scheme (object of class "quad").
- `r` — Optional. Vector of values of the argument r at which the function $G(r)$ should be computed. This argument is usually not specified. There is a sensible default.
- `breaks` — This argument is for internal use only.
- `correction` — Edge correction(s) to be employed in calculating the compensator. Options are "border", "Hanisch" and "best". Alternatively `correction="all"` selects all options.
- `conditional` — Optional. Logical value indicating whether to compute the estimates for the conditional case. See Details.
- `restrict` — Logical value indicating whether to compute the restriction estimator (`restrict=TRUE`) or the reweighting estimator (`restrict=FALSE`, the default). Applies only if `conditional=TRUE`. See Details.
- `model` — Optional. A fitted point process model (object of class "ppm") to be re-fitted to the data using `update.ppm`, if `object` is a point pattern. Overrides the arguments `trend,interaction,rbord,ppmcorrection`.
- `trend,interaction,rbord` — Optional. Arguments passed to `ppm` to fit a point process model to the data, if `object` is a point pattern. See `ppm` for details.
- `...` — Extra arguments passed to `ppm`.
- `ppmcorrection` — The `correction` argument to `ppm`.
- `truecoef` — Optional. Numeric vector. If present, this will be treated as if it were the true coefficient vector of the point process model, in calculating the diagnostic. Incompatible with `hi.res`.

`hi.res` Optional. List of parameters passed to quadscheme. If this argument is present, the model will be re-fitted at high resolution as specified by these parameters. The coefficients of the resulting fitted model will be taken as the true coefficients. Then the diagnostic will be computed for the default quadrature scheme, but using the high resolution coefficients.

Details

This command provides a diagnostic for the goodness-of-fit of a point process model fitted to a point pattern dataset. It computes different estimates of the nearest neighbour distance distribution function G of the dataset, which should be approximately equal if the model is a good fit to the data.

The first argument, `object`, is usually a fitted point process model (object of class `"ppm"`), obtained from the model-fitting function `ppm`.

For convenience, `object` can also be a point pattern (object of class `"ppp"`). In that case, a point process model will be fitted to it, by calling `ppm` using the arguments `trend` (for the first order trend), `interaction` (for the interpoint interaction) and `rbord` (for the erosion distance in the border correction for the pseudolikelihood). See `ppm` for details of these arguments.

The algorithm first extracts the original point pattern dataset (to which the model was fitted) and computes the standard nonparametric estimates of the G function. It then also computes the *model-compensated* G function. The different functions are returned as columns in a data frame (of class `"fv"`). The interpretation of the columns is as follows (ignoring edge corrections):

`bord`: the nonparametric border-correction estimate of $G(r)$,

$$\hat{G}(r) = \frac{\sum_i I\{d_i \le r\} I\{b_i > r\}}{\sum_i I\{b_i > r\}}$$

where d_i is the distance from the i-th data point to its nearest neighbour, and b_i is the distance from the i-th data point to the boundary of the window W.

`bcom`: the model compensator of the border-correction estimate

$$\mathbf{C}\,\hat{G}(r) = \frac{\int \lambda(u,x) I\{b(u) > r\} I\{d(u,x) \le r\}}{1 + \sum_i I\{b_i > r\}}$$

where $\lambda(u,x)$ denotes the conditional intensity of the model at the location u, and $d(u,x)$ denotes the distance from u to the nearest point in x, while $b(u)$ denotes the distance from u to the boundary of the window$W$.

`han`: the nonparametric Hanisch estimate of $G(r)$

$$\hat{G}(r) = \frac{D(r)}{D(\infty)}$$

where

$$D(r) = \sum_i \frac{I\{x_i \in W_{\ominus d_i}\} I\{d_i \le r\}}{\text{area}(W_{\ominus d_i})}$$

in which $W_{\ominus r}$ denotes the erosion of the window W by a distance r.

`hcom`: the corresponding model-compensated function

$$\mathbf{C}\,G(r) = \int_W \frac{\lambda(u,x) I(u \in W_{\ominus d(u)}) I(d(u) \le r)}{\hat{D}(\infty)\text{area}(W_{\ominus d(u)}) + 1}$$

where $d(u) = d(u,x)$ is the ('empty space') distance from location u to the nearest point of x.

If the fitted model is a Poisson point process, then the formulae above are exactly what is computed. If the fitted model is not Poisson, the formulae above are modified slightly to handle edge effects.

The modification is determined by the arguments `conditional` and `restrict`. The value of `conditional` defaults to `FALSE` for Poisson models and `TRUE` for non-Poisson models. If `conditional=FALSE` then the formulae above are not modified. If `conditional=TRUE`, then the algorithm calculates the *restriction estimator* if `restrict=TRUE`, and calculates the *reweighting estimator* if `restrict=FALSE`. See Appendix E of Baddeley, Rubak and Møller (2011). See also `spatstat.options('eroded.intensity')`. Thus, by default, the reweighting estimator is computed for non-Poisson models.

The border-corrected and Hanisch-corrected estimates of $G(r)$ are approximately unbiased estimates of the G-function, assuming the point process is stationary. The model-compensated functions are unbiased estimates *of the mean value of the corresponding nonparametric estimate*, assuming the model is true. Thus, if the model is a good fit, the mean value of the difference between the nonparametric and model-compensated estimates is approximately zero.

To compute the difference between the nonparametric and model-compensated functions, use `Gres`.

Value

A function value table (object of class `"fv"`), essentially a data frame of function values. There is a plot method for this class. See `fv.object`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

Ege Rubak `<rubak@math.aau.dk>` and Jesper Møller.

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Related functions: `Gest`, `Gres`.

Alternative functions: `Kcom`, `psstA`, `psstG`, `psst`.

Model fitting: `ppm`.

Examples

```
    data(cells)
    fit0 <- ppm(cells, ~1) # uniform Poisson
    G0 <- Gcom(fit0)
    G0
    plot(G0)
# uniform Poisson is clearly not correct

# Hanisch estimates only
    plot(Gcom(fit0), cbind(han, hcom) ~ r)

    fit1 <- ppm(cells, ~1, Strauss(0.08))
    plot(Gcom(fit1), cbind(han, hcom) ~ r)

# Try adjusting interaction distance
```

```
fit2 <- update(fit1, Strauss(0.10))
plot(Gcom(fit2), cbind(han, hcom) ~ r)

G3 <- Gcom(cells, interaction=Strauss(0.12))
plot(G3, cbind(han, hcom) ~ r)
```

Gcross *Multitype Nearest Neighbour Distance Function (i-to-j)*

Description

For a multitype point pattern, estimate the distribution of the distance from a point of type i to the nearest point of type j.

Usage

```
Gcross(X, i, j, r=NULL, breaks=NULL, ..., correction=c("rs", "km", "han"))
```

Arguments

X	The observed point pattern, from which an estimate of the cross type distance distribution function $G_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
r	Optional. Numeric vector. The values of the argument r at which the distribution function $G_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
...	Ignored.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "hanisch" and "best". Alternatively correction="all" selects all options.

Details

This function Gcross and its companions Gdot and Gmulti are generalisations of the function Gest to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible "colours" or "types". In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor. The arguments i and j will be interpreted as levels of the factor X$marks. (Warning: this means that an integer value i=3 will be interpreted as the number 3, **not** the 3rd smallest level).

The "cross-type" (type i to type j) nearest neighbour distance distribution function of a multitype point process is the cumulative distribution function $G_{ij}(r)$ of the distance from a typical random point of the process with type i the nearest point of type j.

An estimate of $G_{ij}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the process of type i points were independent of the process of type j points, then $G_{ij}(r)$ would equal $F_j(r)$, the empty space function of the type j points. For a multitype Poisson point process where the type i points have intensity λ_i, we have

$$G_{ij}(r) = 1 - e^{-\lambda_j \pi r^2}$$

Deviations between the empirical and theoretical G_{ij} curves may suggest dependence between the points of types i and j.

This algorithm estimates the distribution function $G_{ij}(r)$ from the point pattern X. It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in Gest.

The argument r is the vector of values for the distance r at which $G_{ij}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of hist) for the computation of histograms of distances. The reduced-sample and Kaplan-Meier estimators are computed from histogram counts. In the case of the Kaplan-Meier estimator this introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify r. However, if it is specified, r must satisfy r[1] = 0, and max(r) must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of r must be finely spaced.

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $G_{ij}(r)$. This estimate should be used with caution as $G_{ij}(r)$ is not necessarily differentiable.

The naive empirical distribution of distances from each point of the pattern X to the nearest other point of the pattern, is a biased estimate of G_{ij}. However this is also returned by the algorithm, as it is sometimes useful in other contexts. Care should be taken not to use the uncorrected empirical G_{ij} as if it were an unbiased estimator of G_{ij}.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing six numeric columns

r	the values of the argument r at which the function $G_{ij}(r)$ has been estimated
rs	the "reduced sample" or "border correction" estimator of $G_{ij}(r)$
han	the Hanisch-style estimator of $G_{ij}(r)$
km	the spatial Kaplan-Meier estimator of $G_{ij}(r)$
hazard	the hazard rate $\lambda(r)$ of $G_{ij}(r)$ by the spatial Kaplan-Meier method
raw	the uncorrected estimate of $G_{ij}(r)$, i.e. the empirical distribution of the distances from each point of type i to the nearest point of type j
theo	the theoretical value of $G_{ij}(r)$ for a marked Poisson process with the same estimated intensity (see below).

Warnings

The arguments `i` and `j` are always interpreted as levels of the factor `X$marks`. They are converted to character strings if they are not already character strings. The value `i=1` does **not** refer to the first level of the factor.

The function G_{ij} does not necessarily have a density.

The reduced sample estimator of G_{ij} is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r. Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of G_{ij} is always nondecreasing but its maximum value may be less than 1.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.

Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

`Gdot`, `Gest`, `Gmulti`

Examples

```
    # amacrine cells data
    G01 <- Gcross(amacrine)

    # equivalent to:
    ## Not run:
    G01 <- Gcross(amacrine, "off", "on")

## End(Not run)

    plot(G01)

    # empty space function of `on' points
    ## Not run:
```

```
        F1 <- Fest(split(amacrine)$on, r = G01$r)
        lines(F1$r, F1$km, lty=3)

## End(Not run)

    # synthetic example
    pp <- runifpoispp(30)
    pp <- pp %mark% factor(sample(0:1, npoints(pp), replace=TRUE))
    G <- Gcross(pp, "0", "1")   # note: "0" not 0
```

Gdot *Multitype Nearest Neighbour Distance Function (i-to-any)*

Description

For a multitype point pattern, estimate the distribution of the distance from a point of type i to the nearest other point of any type.

Usage

```
Gdot(X, i, r=NULL, breaks=NULL, ..., correction=c("km", "rs", "han"))
```

Arguments

X	The observed point pattern, from which an estimate of the distance distribution function $G_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
r	Optional. Numeric vector. The values of the argument r at which the distribution function $G_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
...	Ignored.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "hanisch" and "best". Alternatively correction="all" selects all options.

Details

This function Gdot and its companions Gcross and Gmulti are generalisations of the function Gest to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible "colours" or "types". In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor. The argument will be interpreted as a level of the factor `X$marks`. (Warning: this means that an integer value `i=3` will be interpreted as the number 3, **not** the 3rd smallest level.)

The "dot-type" (type i to any type) nearest neighbour distance distribution function of a multitype point process is the cumulative distribution function $G_{i\bullet}(r)$ of the distance from a typical random point of the process with type i the nearest other point of the process, regardless of type.

An estimate of $G_{i\bullet}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the type i points were independent of all other points, then $G_{i\bullet}(r)$ would equal $G_{ii}(r)$, the nearest neighbour distance distribution function of the type i points alone. For a multitype Poisson point process with total intensity λ, we have

$$G_{i\bullet}(r) = 1 - e^{-\lambda \pi r^2}$$

Deviations between the empirical and theoretical $G_{i\bullet}$ curves may suggest dependence of the type i points on the other points.

This algorithm estimates the distribution function $G_{i\bullet}(r)$ from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in `Gest`.

The argument `r` is the vector of values for the distance r at which $G_{i\bullet}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of `hist`) for the computation of histograms of distances. The reduced-sample and Kaplan-Meier estimators are computed from histogram counts. In the case of the Kaplan-Meier estimator this introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify `r`. However, if it is specified, `r` must satisfy `r[1] = 0`, and `max(r)` must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of `r` must be finely spaced.

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $G_{i\bullet}(r)$. This estimate should be used with caution as $G_{i\bullet}(r)$ is not necessarily differentiable.

The naive empirical distribution of distances from each point of the pattern `X` to the nearest other point of the pattern, is a biased estimate of $G_{i\bullet}$. However this is also returned by the algorithm, as it is sometimes useful in other contexts. Care should be taken not to use the uncorrected empirical $G_{i\bullet}$ as if it were an unbiased estimator of $G_{i\bullet}$.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing six numeric columns

`r`	the values of the argument r at which the function $G_{i\bullet}(r)$ has been estimated
`rs`	the "reduced sample" or "border correction" estimator of $G_{i\bullet}(r)$
`han`	the Hanisch-style estimator of $G_{i\bullet}(r)$
`km`	the spatial Kaplan-Meier estimator of $G_{i\bullet}(r)$
`hazard`	the hazard rate $\lambda(r)$ of $G_{i\bullet}(r)$ by the spatial Kaplan-Meier method
`raw`	the uncorrected estimate of $G_{i\bullet}(r)$, i.e. the empirical distribution of the distances from each point of type i to the nearest other point of any type.
`theo`	the theoretical value of $G_{i\bullet}(r)$ for a marked Poisson process with the same estimated intensity (see below).

Warnings

The argument `i` is interpreted as a level of the factor `X$marks`. It is converted to a character string if it is not already a character string. The value `i=1` does **not** refer to the first level of the factor.

The function $G_{i\bullet}$ does not necessarily have a density.

The reduced sample estimator of $G_{i\bullet}$ is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r. Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of $G_{i\bullet}$ is always nondecreasing but its maximum value may be less than 1.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.

Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

`Gcross`, `Gest`, `Gmulti`

Examples

```
# amacrine cells data
G0. <- Gdot(amacrine, "off")
plot(G0.)

# synthetic example
pp <- runifpoispp(30)
pp <- pp %mark% factor(sample(0:1, npoints(pp), replace=TRUE))
G <- Gdot(pp, "0")
G <- Gdot(pp, 0) # equivalent
```

Gest *Nearest Neighbour Distance Function G*

Description

Estimates the nearest neighbour distance distribution function $G(r)$ from a point pattern in a window of arbitrary shape.

Usage

```
Gest(X, r=NULL, breaks=NULL, ...,
     correction=c("rs", "km", "han"),
     domain=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of $G(r)$ will be computed. An object of class ppp, or data in any format acceptable to as.ppp().
r	Optional. Numeric vector. The values of the argument r at which $G(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
...	Ignored.
correction	Optional. The edge correction(s) to be used to estimate $G(r)$. A vector of character strings selected from "none", "rs", "km", "Hanisch" and "best". Alternatively correction="all" selects all options.
domain	Optional. Calculations will be restricted to this subset of the window. See Details.

Details

The nearest neighbour distance distribution function (also called the "*event-to-event*" or "*inter-event*" distribution) of a point process X is the cumulative distribution function G of the distance from a typical random point of X to the nearest other point of X.

An estimate of G derived from a spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern (Cressie, 1991; Diggle, 1983; Ripley, 1988). In exploratory analyses, the estimate of G is a useful statistic summarising one aspect of the "clustering" of points. For inferential purposes, the estimate of G is usually compared to the true value of G for a completely random (Poisson) point process, which is

$$G(r) = 1 - e^{-\lambda \pi r^2}$$

where λ is the intensity (expected number of points per unit area). Deviations between the empirical and theoretical G curves may suggest spatial clustering or spatial regularity.

This algorithm estimates the nearest neighbour distance distribution function G from the point pattern X. It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape.

The argument `X` is interpreted as a point pattern object (of class `"ppp"`, see `ppp.object`) and can be supplied in any of the formats recognised by `as.ppp()`.

The estimation of G is hampered by edge effects arising from the unobservability of points of the random pattern outside the window. An edge correction is needed to reduce bias (Baddeley, 1998; Ripley, 1988). The edge corrections implemented here are the border method or "*reduced sample*" estimator, the spatial Kaplan-Meier estimator (Baddeley and Gill, 1997) and the Hanisch estimator (Hanisch, 1984).

The argument `r` is the vector of values for the distance r at which $G(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of `hist`) for the computation of histograms of distances. The estimators are computed from histogram counts. This introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify `r`. However, if it is specified, `r` must satisfy `r[1] = 0`, and `max(r)` must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of `r` must be finely spaced.

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $G(r)$. The hazard rate is defined as the derivative

$$\lambda(r) = -\frac{d}{dr}\log(1 - G(r))$$

This estimate should be used with caution as G is not necessarily differentiable.

If the argument `domain` is given, the estimate of $G(r)$ will be based only on the nearest neighbour distances measured from points falling inside `domain` (although their nearest neighbours may lie outside `domain`). This is useful in bootstrap techniques. The argument `domain` should be a window (object of class `"owin"`) or something acceptable to `as.owin`. It must be a subset of the window of the point pattern `X`.

The naive empirical distribution of distances from each point of the pattern `X` to the nearest other point of the pattern, is a biased estimate of G. However it is sometimes useful. It can be returned by the algorithm, by selecting `correction="none"`. Care should be taken not to use the uncorrected empirical G as if it were an unbiased estimator of G.

To simply compute the nearest neighbour distance for each point in the pattern, use `nndist`. To determine which point is the nearest neighbour of a given point, use `nnwhich`.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing some or all of the following columns:

`r`	the values of the argument r at which the function $G(r)$ has been estimated
`rs`	the "reduced sample" or "border correction" estimator of $G(r)$
`km`	the spatial Kaplan-Meier estimator of $G(r)$
`hazard`	the hazard rate $\lambda(r)$ of $G(r)$ by the spatial Kaplan-Meier method
`raw`	the uncorrected estimate of $G(r)$, i.e. the empirical distribution of the distances from each point in the pattern `X` to the nearest other point of the pattern
`han`	the Hanisch correction estimator of $G(r)$
`theo`	the theoretical value of $G(r)$ for a stationary Poisson process of the same estimated intensity.

Warnings

The function G does not necessarily have a density. Any valid c.d.f. may appear as the nearest neighbour distance distribution function of a stationary point process.

The reduced sample estimator of G is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r. Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of G is always nondecreasing but its maximum value may be less than 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37-78.

Baddeley, A.J. and Gill, R.D. Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25** (1997) 263-292.

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Hanisch, K.-H. (1984) Some remarks on estimators of the distribution function of nearest-neighbour distance in stationary spatial point patterns. *Mathematische Operationsforschung und Statistik, series Statistics* **15**, 409–412.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

nndist, nnwhich, Fest, Jest, Kest, km.rs, reduced.sample, kaplan.meier

Examples

```
data(cells)
G <- Gest(cells)
plot(G)

# P-P style plot
plot(G, cbind(km,theo) ~ theo)

# the empirical G is below the Poisson G,
# indicating an inhibited pattern

## Not run:
   plot(G, . ~ r)
   plot(G, . ~ theo)
   plot(G, asin(sqrt(.)) ~ asin(sqrt(theo)))

## End(Not run)
```

Geyer — *Geyer's Saturation Point Process Model*

Description

Creates an instance of Geyer's saturation point process model which can then be fitted to point pattern data.

Usage

```
Geyer(r,sat)
```

Arguments

r	Interaction radius. A positive real number.
sat	Saturation threshold. A non-negative real number.

Details

Geyer (1999) introduced the "saturation process", a modification of the Strauss process (see `Strauss`) in which the total contribution to the potential from each point (from its pairwise interaction with all other points) is trimmed to a maximum value s. The interaction structure of this model is implemented in the function `Geyer()`.

The saturation point process with interaction radius r, saturation threshold s, and parameters β and γ, is the point process in which each point x_i in the pattern X contributes a factor

$$\beta\gamma^{\min(s,t(x_i,X))}$$

to the probability density of the point pattern, where $t(x_i, X)$ denotes the number of 'close neighbours' of x_i in the pattern X. A close neighbour of x_i is a point x_j with $j \neq i$ such that the distance between x_i and x_j is less than or equal to r.

If the saturation threshold s is set to infinity, this model reduces to the Strauss process (see `Strauss`) with interaction parameter γ^2. If $s = 0$, the model reduces to the Poisson point process. If s is a finite positive number, then the interaction parameter γ may take any positive value (unlike the case of the Strauss process), with values $\gamma < 1$ describing an 'ordered' or 'inhibitive' pattern, and values $\gamma > 1$ describing a 'clustered' or 'attractive' pattern.

The nonstationary saturation process is similar except that the value β is replaced by a function $\beta(x_i)$ of location.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the saturation process interaction is yielded by `Geyer(r, sat)` where the arguments `r` and `sat` specify the Strauss interaction radius r and the saturation threshold s, respectively. See the examples below.

Note the only arguments are the interaction radius `r` and the saturation threshold `sat`. When `r` and `sat` are fixed, the model becomes an exponential family. The canonical parameters $\log(\beta)$ and $\log(\gamma)$ are estimated by `ppm()`, not fixed in `Geyer()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of Geyer's saturation point process with interaction radius r and saturation threshold `sat`.

Zero saturation

The value `sat=0` is permitted by `Geyer`, but this is not very useful. For technical reasons, when `ppm` fits a Geyer model with `sat=0`, the default behaviour is to return an "invalid" fitted model in which the estimate of γ is `NA`. In order to get a Poisson process model returned when `sat=0`, you would need to set `emend=TRUE` in the call to `ppm`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

`ppm`, `pairwise.family`, `ppm.object`, `Strauss`, `SatPiece`

Examples

```
ppm(cells, ~1, Geyer(r=0.07, sat=2))
# fit the stationary saturation process to `cells'
```

`Gfox` *Foxall's Distance Functions*

Description

Given a point pattern `X` and a spatial object `Y`, compute estimates of Foxall's G and J functions.

Usage

```
Gfox(X, Y, r = NULL, breaks = NULL, correction = c("km", "rs", "han"), ...)
Jfox(X, Y, r = NULL, breaks = NULL, correction = c("km", "rs", "han"), ...)
```

Arguments

`X`	A point pattern (object of class `"ppp"`) from which distances will be measured.
`Y`	An object of class `"ppp"`, `"psp"` or `"owin"` to which distances will be measured.
`r`	Optional. Numeric vector. The values of the argument r at which $Gfox(r)$ or $Jfox(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`breaks`	This argument is for internal use only.
`correction`	Optional. The edge correction(s) to be used to estimate $Gfox(r)$ or $Jfox(r)$. A vector of character strings selected from `"none"`, `"rs"`, `"km"`, `"cs"` and `"best"`. Alternatively `correction="all"` selects all options.
`...`	Extra arguments affecting the discretisation of distances. These arguments are ignored by `Gfox`, but `Jfox` passes them to `Hest` to determine the discretisation of the spatial domain.

Details

Given a point pattern `X` and another spatial object `Y`, these functions compute two nonparametric measures of association between `X` and `Y`, introduced by Foxall (Foxall and Baddeley, 2002).

Let the random variable R be the distance from a typical point of `X` to the object `Y`. Foxall's G-function is the cumulative distribution function of R:

$$G(r) = P(R \leq r)$$

Let the random variable S be the distance from a *fixed* point in space to the object `Y`. The cumulative distribution function of S is the (unconditional) spherical contact distribution function

$$H(r) = P(S \leq r)$$

which is computed by `Hest`.

Foxall's J-function is the ratio

$$J(r) = \frac{1 - G(r)}{1 - H(r)}$$

For further interpretation, see Foxall and Baddeley (2002).

Accuracy of `Jfox` depends on the pixel resolution, which is controlled by the arguments `eps`, `dimyx` and `xy` passed to `as.mask`. For example, use `eps=0.1` to specify square pixels of side 0.1 units, and `dimyx=256` to specify a 256 by 256 grid of pixels.

Value

A function value table (object of class `"fv"`) which can be printed, plotted, or converted to a data frame of values.

Author(s)

Rob Foxall and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Foxall, R. and Baddeley, A. (2002) Nonparametric measures of association between a spatial point process and a random set, with geological applications. *Applied Statistics* **51**, 165–182.

See Also

`Gest`, `Hest`, `Jest`, `Fest`

Examples

```
  data(copper)
  X <- copper$SouthPoints
  Y <- copper$SouthLines
  G <- Gfox(X,Y)
  J <- Jfox(X,Y, correction="km")

  ## Not run:
  J <- Jfox(X,Y, correction="km", eps=0.25)

## End(Not run)
```

Ginhom *Inhomogeneous Nearest Neighbour Function*

Description

Estimates the inhomogeneous nearest neighbour function G of a non-stationary point pattern.

Usage

```
Ginhom(X, lambda = NULL, lmin = NULL, ...,
      sigma = NULL, varcov = NULL,
      r = NULL, breaks = NULL, ratio = FALSE, update = TRUE)
```

Arguments

X	The observed data point pattern, from which an estimate of the inhomogeneous G function will be computed. An object of class "ppp" or in a format recognised by as.ppp()
lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
lmin	Optional. The minimum possible value of the intensity over the spatial domain. A positive numerical value.
sigma,varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.
...	Extra arguments passed to as.mask to control the pixel resolution, or passed to density.ppp to control the smoothing bandwidth.
r	vector of values for the argument r at which the inhomogeneous K function should be evaluated. Not normally given by the user; there is a sensible default.
breaks	This argument is for internal use only.
ratio	Logical. If TRUE, the numerator and denominator of the estimate will also be saved, for use in analysing replicated point patterns.
update	Logical. If lambda is a fitted model (class "ppm" or "kppm") and update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without fitting it to X.

Details

This command computes estimates of the inhomogeneous G-function (van Lieshout, 2010) of a point pattern. It is the counterpart, for inhomogeneous spatial point patterns, of the nearest-neighbour distance distribution function G for homogeneous point patterns computed by Gest.

The argument X should be a point pattern (object of class "ppp").

The inhomogeneous G function is computed using the border correction, equation (7) in Van Lieshout (2010).

The argument lambda should supply the (estimated) values of the intensity function λ of the point process. It may be either

a numeric vector containing the values of the intensity function at the points of the pattern `X`.

a pixel image (object of class `"im"`) assumed to contain the values of the intensity function at all locations in the window.

a fitted point process model (object of class `"ppm"` or `"kppm"`) whose fitted *trend* can be used as the fitted intensity. (If `update=TRUE` the model will first be refitted to the data `X` before the trend is computed.)

a function which can be evaluated to give values of the intensity at any locations.

omitted: if `lambda` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

If `lambda` is a numeric vector, then its length should be equal to the number of points in the pattern `X`. The value `lambda[i]` is assumed to be the the (estimated) value of the intensity $\lambda(x_i)$ for the point x_i of the pattern X. Each value must be a positive number; `NA`'s are not allowed.

If `lambda` is a pixel image, the domain of the image should cover the entire window of the point pattern. If it does not (which may occur near the boundary because of discretisation error), then the missing pixel values will be obtained by applying a Gaussian blur to `lambda` using `blur`, then looking up the values of this blurred image for the missing locations. (A warning will be issued in this case.)

If `lambda` is a function, then it will be evaluated in the form `lambda(x,y)` where `x` and `y` are vectors of coordinates of the points of `X`. It should return a numeric vector with length equal to the number of points in `X`.

If `lambda` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate `lambda[i]` for the point `X[i]` is computed by removing `X[i]` from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point `X[i]`. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Author(s)

Original code by Marie-Colette van Lieshout. C implementation and R adaptation by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Møller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.

Van Lieshout, M.N.M. (2010) A J-function for inhomogeneous point processes. *Statistica Neerlandica* **65**, 183–201.

See Also

`Finhom`, `Jinhom`, `Gest`

Examples

```
## Not run:
  plot(Ginhom(swedishpines, sigma=bw.diggle, adjust=2))

## End(Not run)
  plot(Ginhom(swedishpines, sigma=10))
```

Gmulti *Marked Nearest Neighbour Distance Function*

Description

For a marked point pattern, estimate the distribution of the distance from a typical point in subset I to the nearest point of subset J.

Usage

```
Gmulti(X, I, J, r=NULL, breaks=NULL, ...,
        disjoint=NULL, correction=c("rs", "km", "han"))
```

Arguments

X	The observed point pattern, from which an estimate of the multitype distance distribution function $G_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
I	Subset of points of X from which distances are measured.
J	Subset of points in X to which distances are measured.
r	Optional. Numeric vector. The values of the argument r at which the distribution function $G_{IJ}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
...	Ignored.
disjoint	Optional flag indicating whether the subsets I and J are disjoint. If missing, this value will be computed by inspecting the vectors I and J.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "hanisch" and "best". Alternatively correction="all" selects all options.

Details

The function Gmulti generalises Gest (for unmarked point patterns) and Gdot and Gcross (for multitype point patterns) to arbitrary marked point patterns.

Suppose X_I, X_J are subsets, possibly overlapping, of a marked point process. This function computes an estimate of the cumulative distribution function $G_{IJ}(r)$ of the distance from a typical point of X_I to the nearest distinct point of X_J.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp.

The arguments I and J specify two subsets of the point pattern. They may be any type of subset indices, for example, logical vectors of length equal to npoints(X), or integer vectors with entries in the range 1 to npoints(X), or negative integer vectors.

Alternatively, I and J may be **functions** that will be applied to the point pattern X to obtain index vectors. If I is a function, then evaluating I(X) should yield a valid subset index. This option is useful when generating simulation envelopes using envelope.

This algorithm estimates the distribution function $G_{IJ}(r)$ from the point pattern X. It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in Gest.

The argument r is the vector of values for the distance r at which $G_{IJ}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of hist) for the computation of histograms of distances. The reduced-sample and Kaplan-Meier estimators are computed from histogram counts. In the case of the Kaplan-Meier estimator this introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify r. However, if it is specified, r must satisfy r[1] = 0, and max(r) must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of r must be finely spaced.

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $G_{IJ}(r)$. This estimate should be used with caution as $G_{IJ}(r)$ is not necessarily differentiable.

The naive empirical distribution of distances from each point of the pattern X to the nearest other point of the pattern, is a biased estimate of G_{IJ}. However this is also returned by the algorithm, as it is sometimes useful in other contexts. Care should be taken not to use the uncorrected empirical G_{IJ} as if it were an unbiased estimator of G_{IJ}.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing six numeric columns

r	the values of the argument r at which the function $G_{IJ}(r)$ has been estimated
rs	the "reduced sample" or "border correction" estimator of $G_{IJ}(r)$
han	the Hanisch-style estimator of $G_{IJ}(r)$
km	the spatial Kaplan-Meier estimator of $G_{IJ}(r)$
hazard	the hazard rate $\lambda(r)$ of $G_{IJ}(r)$ by the spatial Kaplan-Meier method
raw	the uncorrected estimate of $G_{IJ}(r)$, i.e. the empirical distribution of the distances from each point of type i to the nearest point of type j
theo	the theoretical value of $G_{IJ}(r)$ for a marked Poisson process with the same estimated intensity

Warnings

The function G_{IJ} does not necessarily have a density.

The reduced sample estimator of G_{IJ} is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r. Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of G_{IJ} is always nondecreasing but its maximum value may be less than 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.

Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

Gcross, Gdot, Gest

Examples

```
trees <- longleaf
 # Longleaf Pine data: marks represent diameter

Gm <- Gmulti(trees, marks(trees) <= 15, marks(trees) >= 25)
plot(Gm)
```

GmultiInhom *Inhomogeneous Marked G-Function*

Description

For a marked point pattern, estimate the inhomogeneous version of the multitype G function, effectively the cumulative distribution function of the distance from a point in subset I to the nearest point in subset J, adjusted for spatially varying intensity.

Usage

```
GmultiInhom(X, I, J,
            lambda = NULL, lambdaI = NULL, lambdaJ = NULL,
            lambdamin = NULL, ...,
            r = NULL,
            ReferenceMeasureMarkSetI = NULL,
            ratio = FALSE)
```

Arguments

- `X` — A spatial point pattern (object of class "ppp".
- `I` — A subset index specifying the subset of points *from* which distances are measured. Any kind of subset index acceptable to `[.ppp`.
- `J` — A subset index specifying the subset of points *to* which distances are measured. Any kind of subset index acceptable to `[.ppp`.
- `lambda` — Intensity estimates for each point of `X`. A numeric vector of length equal to `npoints(X)`. Incompatible with `lambdaI,lambdaJ`.
- `lambdaI` — Intensity estimates for each point of `X[I]`. A numeric vector of length equal to `npoints(X[I])`. Incompatible with `lambda`.
- `lambdaJ` — Intensity estimates for each point of `X[J]`. A numeric vector of length equal to `npoints(X[J])`. Incompatible with `lambda`.
- `lambdamin` — A lower bound for the intensity, or at least a lower bound for the values in `lambdaJ` or `lambda[J]`.
- `...` — Ignored.
- `r` — Vector of distance values at which the inhomogeneous G function should be estimated. There is a sensible default.
- `ReferenceMeasureMarkSetI` — Optional. The total measure of the mark set. A positive number.
- `ratio` — Logical value indicating whether to save ratio information.

Details

See Cronie and Van Lieshout (2015).

Value

Object of class "fv" containing the estimate of the inhomogeneous multitype G function.

Author(s)

Ottmar Cronie and Marie-Colette van Lieshout. Rewritten for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.

References

Cronie, O. and Van Lieshout, M.N.M. (2015) Summary statistics for inhomogeneous marked point processes. *Annals of the Institute of Statistical Mathematics* DOI: 10.1007/s10463-015-0515-z

See Also

`Ginhom`, `Gmulti`

Examples

```
X <- amacrine
I <- (marks(X) == "on")
J <- (marks(X) == "off")
mod <- ppm(X ~ marks * x)
lam <- fitted(mod, dataonly=TRUE)
lmin <- min(predict(mod)[["off"]]) * 0.9
plot(GmultiInhom(X, I, J, lambda=lam, lambdamin=lmin))
```

Gres *Residual G Function*

Description

Given a point process model fitted to a point pattern dataset, this function computes the residual G function, which serves as a diagnostic for goodness-of-fit of the model.

Usage

```
Gres(object, ...)
```

Arguments

object	Object to be analysed. Either a fitted point process model (object of class "ppm"), a point pattern (object of class "ppp"), a quadrature scheme (object of class "quad"), or the value returned by a previous call to Gcom.
...	Arguments passed to Gcom.

Details

This command provides a diagnostic for the goodness-of-fit of a point process model fitted to a point pattern dataset. It computes a residual version of the G function of the dataset, which should be approximately zero if the model is a good fit to the data.

In normal use, object is a fitted point process model or a point pattern. Then Gres first calls Gcom to compute both the nonparametric estimate of the G function and its model compensator. Then Gres computes the difference between them, which is the residual G-function.

Alternatively, object may be a function value table (object of class "fv") that was returned by a previous call to Gcom. Then Gres computes the residual from this object.

Value

A function value table (object of class "fv"), essentially a data frame of function values. There is a plot method for this class. See fv.object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Ege Rubak <rubak@math.aau.dk> and Jesper Møller.

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Related functions: Gcom, Gest.

Alternative functions: Kres, psstA, psstG, psst.

Model-fitting: ppm.

Examples

```
    data(cells)
    fit0 <- ppm(cells, ~1) # uniform Poisson
    G0 <- Gres(fit0)
    plot(G0)
# Hanisch correction estimate
    plot(G0, hres ~ r)
# uniform Poisson is clearly not correct

    fit1 <- ppm(cells, ~1, Strauss(0.08))
    plot(Gres(fit1), hres ~ r)
# fit looks approximately OK; try adjusting interaction distance

    plot(Gres(cells, interaction=Strauss(0.12)))

# How to make envelopes
    ## Not run:
    E <- envelope(fit1, Gres, model=fit1, nsim=39)
    plot(E)

## End(Not run)
# For computational efficiency
    Gc <- Gcom(fit1)
    G1 <- Gres(Gc)
```

`gridcentres` *Rectangular grid of points*

Description

Generates a rectangular grid of points in a window

Usage

```
gridcentres(window, nx, ny)
```

Arguments

`window`	A window. An object of class `owin`, or data in any format acceptable to `as.owin()`.
`nx`	Number of points in each row of the rectangular grid.
`ny`	Number of points in each column of the rectangular grid.

Details

This function creates a rectangular grid of points in the window.

The bounding rectangle of the `window` is divided into a regular $nx \times ny$ grid of rectangular tiles. The function returns the x, y coordinates of the centres of these tiles.

Note that some of these grid points may lie outside the window, if `window` is not of type `"rectangle"`. The function `inside.owin` can be used to select those grid points which do lie inside the window. See the examples.

This function is useful in creating dummy points for quadrature schemes (see `quadscheme`) and for other miscellaneous purposes.

Value

A list with two components x and y, which are numeric vectors giving the coordinates of the points of the rectangular grid.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quad.object, quadscheme, inside.owin, stratrand

Examples

```
  w <- unit.square()
  xy <- gridcentres(w, 10,15)
  ## Not run:
  plot(w)
  points(xy)

## End(Not run)

  bdry <- list(x=c(0.1,0.3,0.7,0.4,0.2),
               y=c(0.1,0.1,0.5,0.7,0.3))
  w <- owin(c(0,1), c(0,1), poly=bdry)
  xy <- gridcentres(w, 30, 30)
  ok <- inside.owin(xy$x, xy$y, w)
  ## Not run:
  plot(w)
  points(xy$x[ok], xy$y[ok])

## End(Not run)
```

---

gridweights *Compute Quadrature Weights Based on Grid Counts*

Description

Computes quadrature weights for a given set of points, using the "counting weights" for a grid of rectangular tiles.

Usage

```
gridweights(X, ntile, ..., window=NULL, verbose=FALSE, npix=NULL, areas=NULL)
```

Arguments

X	Data defining a point pattern.
ntile	Number of tiles in each row and column of the rectangular grid. An integer vector of length 1 or 2.
...	Ignored.

`window`	Default window for the point pattern
`verbose`	Logical flag. If `TRUE`, information will be printed about the computation of the grid weights.
`npix`	Dimensions of pixel grid to use when computing a digital approximation to the tile areas.
`areas`	Vector of areas of the tiles, if they are already known.

Details

This function computes a set of quadrature weights for a given pattern of points (typically comprising both "data" and 'dummy" points). See `quad.object` for an explanation of quadrature weights and quadrature schemes.

The weights are computed by the "counting weights" rule based on a regular grid of rectangular tiles. First `X` and (optionally) `window` are converted into a point pattern object. Then the bounding rectangle of the window of the point pattern is divided into a regular `ntile[1] * ntile[2]` grid of rectangular tiles. The weight attached to a point of `X` is the area of the tile in which it lies, divided by the number of points of `X` lying in that tile.

For non-rectangular windows the tile areas are currently calculated by approximating the window as a binary mask. The accuracy of this approximation is controlled by `npix`, which becomes the argument `dimyx` of `as.mask`.

Value

Vector of nonnegative weights for each point in `X`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`quad.object`, `dirichletWeights`

Examples

```
Q <- quadscheme(runifpoispp(10))
X <- as.ppp(Q) # data and dummy points together
w <- gridweights(X, 10)
w <- gridweights(X, c(10, 10))
```

`grow.boxx` *Add margins to box in any dimension*

Description

Adds a margin to a box of class boxx.

Usage

```
grow.boxx(W, left, right = left)
grow.box3(W, left, right = left)
```

Arguments

W	A box (object of class "boxx" or "box3").
left	Width of margin to be added to left endpoint of box side in every dimension. A single nonnegative number, or a vector of same length as the dimension of the box to add different left margin in each dimension.
right	Width of margin to be added to right endpoint of box side in every dimension. A single nonnegative number, or a vector of same length as the dimension of the box to add different right margin in each dimension.

Value

Another object of the same class "boxx" or "box3" representing the window after margins are added.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

grow.rectangle, boxx, box3

Examples

```
w <- boxx(c(0,10), c(0,10), c(0,10), c(0,10))
# add a margin of size 1 on both sides in all four dimensions
b12 <- grow.boxx(w, 1)

# add margin of size 2 at left, and margin of size 3 at right,
# in each dimension.
v <- grow.boxx(w, 2, 3)
```

---

grow.rectangle *Add margins to rectangle*

Description

Adds a margin to a rectangle.

Usage

```
grow.rectangle(W, xmargin=0, ymargin=xmargin, fraction=NULL)
```

Arguments

W	A window (object of class "owin"). Must be of type "rectangle".
xmargin	Width of horizontal margin to be added. A single nonnegative number, or a vector of length 2 indicating margins of unequal width at left and right.
ymargin	Height of vertical margin to be added. A single nonnegative number, or a vector of length 2 indicating margins of unequal width at bottom and top.
fraction	Fraction of width and height to be added. A number greater than zero, or a numeric vector of length 2 indicating different fractions of width and of height, respectively. Incompatible with specifying xmargin and ymargin.

Details

This is a simple convenience function to add a margin of specified width and height on each side of a rectangular window. Unequal margins can also be added.

Value

Another object of class "owin" representing the window after margins are added.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

trim.rectangle, dilation, erosion, owin.object

Examples

```
w <- square(10)
# add a margin of width 1 on all four sides
square12 <- grow.rectangle(w, 1)

# add margin of width 3 on the right side
# and margin of height 4 on top.
v <- grow.rectangle(w, c(0,3), c(0,4))

# grow by 5 percent on all sides
grow.rectangle(w, fraction=0.05)
```

Hardcore — *The Hard Core Point Process Model*

Description

Creates an instance of the hard core point process model which can then be fitted to point pattern data.

Usage

```
Hardcore(hc=NA)
```

Arguments

hc The hard core distance

Details

A hard core process with hard core distance h and abundance parameter β is a pairwise interaction point process in which distinct points are not allowed to come closer than a distance h apart.

The probability density is zero if any pair of points is closer than h units apart, and otherwise equals

$$f(x_1, \ldots, x_n) = \alpha \beta^{n(x)}$$

where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, and α is the normalising constant.

The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the hard core process pairwise interaction is yielded by the function Hardcore(). See the examples below.

If the hard core distance argument hc is missing or NA, it will be estimated from the data when ppm is called. The estimated value of hc is the minimum nearest neighbour distance multiplied by $n/(n+1)$, where n is the number of data points.

Value

An object of class "interact" describing the interpoint interaction structure of the hard core process with hard core distance hc.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

See Also

Strauss, StraussHard, MultiHard, ppm, pairwise.family, ppm.object

Examples

```
   Hardcore(0.02)
   # prints a sensible description of itself

   ## Not run:
   ppm(cells, ~1, Hardcore(0.05))
   # fit the stationary hard core process to `cells'

## End(Not run)

   # estimate hard core radius from data
   ppm(cells, ~1, Hardcore())
```

```
ppm(cells, ~1, Hardcore)

ppm(cells, ~ polynom(x,y,3), Hardcore(0.05))
# fit a nonstationary hard core process
# with log-cubic polynomial trend
```

---

harmonic *Basis for Harmonic Functions*

Description

Evaluates a basis for the harmonic polynomials in x and y of degree less than or equal to n.

Usage

```
harmonic(x, y, n)
```

Arguments

x	Vector of x coordinates
y	Vector of y coordinates
n	Maximum degree of polynomial

Details

This function computes a basis for the harmonic polynomials in two variables x and y up to a given degree n and evaluates them at given x, y locations. It can be used in model formulas (for example in the model-fitting functions lm,glm,gam and ppm) to specify a linear predictor which is a harmonic function.

A function $f(x, y)$ is harmonic if

$$\frac{\partial^2}{\partial x^2} f + \frac{\partial^2}{\partial y^2} f = 0.$$

The harmonic polynomials of degree less than or equal to n have a basis consisting of $2n$ functions.

This function was implemented on a suggestion of P. McCullagh for fitting nonstationary spatial trend to point process models.

Value

A data frame with `2 * n` columns giving the values of the basis functions at the coordinates. Each column is labelled by an algebraic expression for the corresponding basis function.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

ppm, polynom

Examples

```
# inhomogeneous point pattern
X <- unmark(longleaf)


# fit Poisson point process with log-cubic intensity
fit.3 <- ppm(X ~ polynom(x,y,3), Poisson())

# fit Poisson process with log-cubic-harmonic intensity
fit.h <- ppm(X ~ harmonic(x,y,3), Poisson())

# Likelihood ratio test
lrts <- 2 * (logLik(fit.3) - logLik(fit.h))
df <- with(coords(X),
           ncol(polynom(x,y,3)) - ncol(harmonic(x,y,3)))
pval <- 1 - pchisq(lrts, df=df)
```

---

harmonise *Make Objects Compatible*

Description

Converts several objects of the same class to a common format so that they can be combined or compared.

Usage

```
harmonise(...)
harmonize(...)
```

Arguments

...	Any number of objects of the same class.

Details

This generic command takes any number of objects of the same class, and *attempts* to make them compatible in the sense of `compatible` so that they can be combined or compared.

There are methods for the classes `"fv"` (`harmonise.fv`) and `"im"` (`harmonise.im`).

All arguments `...` must be objects of the same class. The result will be a list, of length equal to the number of arguments `...`, containing new versions of each of these objects, converted to a common format. If the arguments were named (`name=value`) then the return value also carries these names.

Value

A list, of length equal to the number of arguments `...`, whose entries are objects of the same class. If the arguments were named (`name=value`) then the return value also carries these names.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

See Also

compatible, harmonise.fv, harmonise.im

---

harmonise.fv *Make Function Tables Compatible*

Description

Convert several objects of class "fv" to the same values of the function argument.

Usage

```
## S3 method for class 'fv'
harmonise(..., strict=FALSE)

## S3 method for class 'fv'
harmonize(..., strict=FALSE)
```

Arguments

...	Any number of function tables (objects of class "fv").
strict	Logical. If TRUE, a column of data will be deleted if columns of the same name do not appear in every object.

Details

A function value table (object of class "fv") is essentially a data frame giving the values of a function $f(x)$ (or several alternative estimates of this value) at equally-spaced values of the function argument x.

The command harmonise is generic. This is the method for objects of class "fv".

This command makes any number of "fv" objects compatible, in the loose sense that they have the same sequence of values of x. They can then be combined by cbind.fv, but not necessarily by eval.fv.

All arguments ... must be function value tables (objects of class "fv"). The result will be a list, of length equal to the number of arguments ..., containing new versions of each of these functions, converted to a common sequence of x values. If the arguments were named (name=value) then the return value also carries these names.

The range of x values in the resulting functions will be the intersection of the ranges of x values in the original functions. The spacing of x values in the resulting functions will be the finest (narrowest) of the spacings of the x values in the original functions. Function values are interpolated using approxfun.

If strict=TRUE, each column of data will be retained only if a column of the same name appears in all of the arguments .... This ensures that the resulting objects are strictly compatible in the sense of compatible.fv, and can be combined using eval.fv or collapse.fv.

If strict=FALSE (the default), this does not occur, and then the resulting objects are **not** guaranteed to be compatible in the sense of compatible.fv.

Value

A list, of length equal to the number of arguments `...`, whose entries are objects of class `"fv"`. If the arguments were named (`name=value`) then the return value also carries these names.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

See Also

`fv.object`, `cbind.fv`, `eval.fv`, `compatible.fv`

Examples

```
   H <- harmonise(K=Kest(cells), G=Gest(cells))
   H
   ## Not run:
      ## generates a warning about duplicated columns
      try(cbind(H$K, H$G))

## End(Not run)
```

`harmonise.im` *Make Pixel Images Compatible*

Description

Convert several pixel images to a common pixel raster.

Usage

```
## S3 method for class 'im'
harmonise(...)

## S3 method for class 'im'
harmonize(...)
```

Arguments

`...`	Any number of pixel images (objects of class `"im"`) or data which can be converted to pixel images by `as.im`.

Details

This function makes any number of pixel images compatible, by converting them all to a common pixel grid.

The command `harmonise` is generic. This is the method for objects of class `"im"`.

At least one of the arguments `...` must be a pixel image. Some arguments may be windows (objects of class `"owin"`), functions (`function(x,y)`) or numerical constants. These will be converted to images using `as.im`.

The common pixel grid is determined by inspecting all the pixel images in the argument list, computing the bounding box of all the images, then finding the image with the highest spatial resolution, and extending its pixel grid to cover the bounding box.

The return value is a list with entries corresponding to the input arguments. If the arguments were named (`name=value`) then the return value also carries these names.

If you just want to determine the appropriate pixel resolution, without converting the images, use `commonGrid`.

Value

A list, of length equal to the number of arguments `...`, whose entries are pixel images.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`commonGrid`, `compatible.im`, `as.im`

Examples

```
A <- setcov(square(1))
B <- function(x,y) { x }
G <- density(runifpoint(42))
harmonise(X=A, Y=B, Z=G)
```

`harmonise.msr` *Make Measures Compatible*

Description

Convert several measures to a common quadrature scheme

Usage

```
## S3 method for class 'msr'
harmonise(...)
```

Arguments

`...`	Any number of measures (objects of class `"msr"`).

Details

This function makes any number of measures compatible, by converting them all to a common quadrature scheme.

The command harmonise is generic. This is the method for objects of class "msr".

Value

A list, of length equal to the number of arguments ..., whose entries are measures.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

harmonise, msr

Examples

```
fit1 <- ppm(cells ~ x)
fit2 <- ppm(rpoispp(ex=cells) ~ x)
m1 <- residuals(fit1)
m2 <- residuals(fit2)
harmonise(m1, m2)
s1 <- residuals(fit1, type="score")
s2 <- residuals(fit2, type="score")
harmonise(s1, s2)
```

---

harmonise.owin *Make Windows Compatible*

Description

Convert several windows to a common pixel raster.

Usage

```
## S3 method for class 'owin'
harmonise(...)

## S3 method for class 'owin'
harmonize(...)
```

Arguments

...	Any number of windows (objects of class "owin") or data which can be converted to windows by as.owin.

Details

This function makes any number of windows compatible, by converting them all to a common pixel grid.

This only has an effect if one of the windows is a binary mask. If all the windows are rectangular or polygonal, they are returned unchanged.

The command `harmonise` is generic. This is the method for objects of class `"owin"`.

Each argument must be a window (object of class `"owin"`), or data that can be converted to a window by `as.owin`.

The common pixel grid is determined by inspecting all the windows in the argument list, computing the bounding box of all the windows, then finding the binary mask with the finest spatial resolution, and extending its pixel grid to cover the bounding box.

The return value is a list with entries corresponding to the input arguments. If the arguments were named (`name=value`) then the return value also carries these names.

If you just want to determine the appropriate pixel resolution, without converting the windows, use `commonGrid`.

Value

A list of windows, of length equal to the number of arguments `...`. The list belongs to the class `"solist"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`commonGrid`, `harmonise.im`, `as.owin`

Examples

```
harmonise(X=letterR,
          Y=grow.rectangle(Frame(letterR), 0.2),
          Z=as.mask(letterR, eps=0.1),
          V=as.mask(letterR, eps=0.07))
```

`has.close` *Check Whether Points Have Close Neighbours*

Description

For each point in a point pattern, determine whether the point has a close neighbour in the same pattern.

Usage

```
has.close(X, r, Y=NULL, ...)

  ## Default S3 method:
has.close(X,r, Y=NULL, ..., periodic=FALSE)

  ## S3 method for class 'ppp'
has.close(X,r, Y=NULL, ..., periodic=FALSE, sorted=FALSE)

  ## S3 method for class 'pp3'
has.close(X,r, Y=NULL, ..., periodic=FALSE, sorted=FALSE)
```

Arguments

X,Y	Point patterns of class "ppp" or "pp3" or "lpp".
r	Threshold distance: a number greater than zero.
periodic	Logical value indicating whether to measure distances in the periodic sense, so that opposite sides of the (rectangular) window are treated as identical.
sorted	Logical value, indicating whether the points of X (and Y, if given) are already sorted into increasing order of the x coordinates.
...	Other arguments are ignored.

Details

This is simply a faster version of (nndist(X) <= r) or (nncross(X,Y,what="dist") <= r).

has.close(X,r) determines, for each point in the pattern X, whether or not this point has a neighbour in the same pattern X which lies at a distance less than or equal to r.

has.close(X,r,Y) determines, for each point in the pattern X, whether or not this point has a neighbour in the *other* pattern Y which lies at a distance less than or equal to r.

The function has.close is generic, with methods for "ppp" and "pp3" and a default method.

Value

A logical vector, with one entry for each point of X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

nndist

Examples

```
has.close(redwood, 0.05)
with(split(amacrine), has.close(on, 0.05, off))
```

headtail *First or Last Part of a Spatial Pattern*

Description

Returns the first few elements (head) or the last few elements (`tail`) of a spatial pattern.

Usage

```
  ## S3 method for class 'ppp'
head(x, n = 6L, ...)

  ## S3 method for class 'ppx'
head(x, n = 6L, ...)

  ## S3 method for class 'psp'
head(x, n = 6L, ...)

  ## S3 method for class 'tess'
head(x, n = 6L, ...)

  ## S3 method for class 'ppp'
tail(x, n = 6L, ...)

  ## S3 method for class 'ppx'
tail(x, n = 6L, ...)

  ## S3 method for class 'psp'
tail(x, n = 6L, ...)

  ## S3 method for class 'tess'
tail(x, n = 6L, ...)
```

Arguments

`x`	A spatial pattern of geometrical figures, such as a spatial pattern of points (an object of class `"ppp"`, `"pp3"`, `"ppx"` or `"lpp"`) or a spatial pattern of line segments (an object of class `"psp"`) or a tessellation (object of class `"tess"`).
`n`	Integer. The number of elements of the pattern that should be extracted.
`...`	Ignored.

Details

These are methods for the generic functions `head` and `tail`. They extract the first or last `n` elements from `x` and return them as an object of the same kind as `x`.

To inspect the spatial coordinates themselves, use `View(x)` or `head(as.data.frame(x))`.

Value

An object of the same class as `x`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

View, edit.

Conversion to data frame: as.data.frame.ppp, as.data.frame.ppx, as.data.frame.psp

Examples

```
head(cells)
tail(as.psp(spiders), 10)
head(dirichlet(cells), 4)
```

Hest *Spherical Contact Distribution Function*

Description

Estimates the spherical contact distribution function of a random set.

Usage

```
Hest(X, r=NULL, breaks=NULL, ...,
     W,
     correction=c("km", "rs", "han"),
     conditional=TRUE)
```

Arguments

X	The observed random set. An object of class "ppp", "psp" or "owin". Alternatively a pixel image (class "im") with logical values.
r	Optional. Vector of values for the argument r at which $H(r)$ should be evaluated. Users are advised *not* to specify this argument; there is a sensible default.
breaks	This argument is for internal use only.
...	Arguments passed to as.mask to control the discretisation.
W	Optional. A window (object of class "owin") to be taken as the window of observation. The contact distribution function will be estimated from values of the contact distance inside W.
correction	Optional. The edge correction(s) to be used to estimate $H(r)$. A vector of character strings selected from "none", "rs", "km", "han" and "best". Alternatively correction="all" selects all options.
conditional	Logical value indicating whether to compute the conditional or unconditional distribution. See Details.

Details

The spherical contact distribution function of a stationary random set X is the cumulative distribution function H of the distance from a fixed point in space to the nearest point of X, given that the point lies outside X. That is, $H(r)$ equals the probability that X lies closer than r units away from the fixed point x, given that X does not cover x.

Let $D = d(x, X)$ be the shortest distance from an arbitrary point x to the set X. Then the spherical contact distribution function is

$$H(r) = P(D \le r \mid D > 0)$$

For a point process, the spherical contact distribution function is the same as the empty space function F discussed in Fest.

The argument X may be a point pattern (object of class "ppp"), a line segment pattern (object of class "psp") or a window (object of class "owin"). It is assumed to be a realisation of a stationary random set.

The algorithm first calls distmap to compute the distance transform of X, then computes the Kaplan-Meier and reduced-sample estimates of the cumulative distribution following Hansen et al (1999). If conditional=TRUE (the default) the algorithm returns an estimate of the spherical contact function $H(r)$ as defined above. If conditional=FALSE, it instead returns an estimate of the cumulative distribution function $H^*(r) = P(D \le r)$ which includes a jump at $r = 0$ if X has nonzero area.

Accuracy depends on the pixel resolution, which is controlled by the arguments eps, dimyx and xy passed to as.mask. For example, use eps=0.1 to specify square pixels of side 0.1 units, and dimyx=256 to specify a 256 by 256 grid of pixels.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Essentially a data frame containing up to six columns:

r	the values of the argument r at which the function $H(r)$ has been estimated
rs	the "reduced sample" or "border correction" estimator of $H(r)$
km	the spatial Kaplan-Meier estimator of $H(r)$
hazard	the hazard rate $\lambda(r)$ of $H(r)$ by the spatial Kaplan-Meier method
han	the spatial Hanisch-Chiu-Stoyan estimator of $H(r)$
raw	the uncorrected estimate of $H(r)$, i.e. the empirical distribution of the distance from a fixed point in the window to the nearest point of X

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> with contributions from Kassel Hingee.

References

Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37-78.

Baddeley, A.J. and Gill, R.D. The empty space hazard of a spatial pattern. Research Report 1994/3, Department of Mathematics, University of Western Australia, May 1994.

Hansen, M.B., Baddeley, A.J. and Gill, R.D. First contact distributions for spatial patterns: regularity and estimation. *Advances in Applied Probability* **31** (1999) 15-33.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

Fest

Examples

```
X <- runifpoint(42)
H <- Hest(X)
Y <- rpoisline(10)
H <- Hest(Y)
H <- Hest(Y, dimyx=256)
X <- heather$coarse
plot(Hest(X))
H <- Hest(X, conditional=FALSE)

P <- owin(poly=list(x=c(5.3, 8.5, 8.3, 3.7, 1.3, 3.7),
                    y=c(9.7, 10.0, 13.6, 14.4, 10.7, 7.2)))
plot(X)
plot(P, add=TRUE, col="red")
H <- Hest(X, W=P)
Z <- as.im(FALSE, Frame(X))
Z[X] <- TRUE
Z <- Z[P, drop=FALSE]
plot(Z)
H <- Hest(Z)
```

---

hextess *Hexagonal Grid or Tessellation*

Description

Construct a hexagonal grid of points, or a hexagonal tessellation.

Usage

```
hexgrid(W, s, offset = c(0, 0), origin=NULL, trim = TRUE)

hextess(W, s, offset = c(0, 0), origin=NULL, trim = TRUE)
```

Arguments

W	Window in which to construct the hexagonal grid or tessellation. An object of class "owin".
s	Side length of hexagons. A positive number.
offset	Numeric vector of length 2 specifying a shift of the hexagonal grid. See Details.

`origin`	Numeric vector of length 2 specifying the initial origin of the hexagonal grid, before the offset is applied. See Details.
`trim`	Logical value indicating whether to restrict the result to the window `W`. See Details.

Details

`hexgrid` constructs a hexagonal grid of points on the window `W`. If `trim=TRUE` (the default), the grid is intersected with `W` so that all points lie inside `W`. If `trim=FALSE`, then we retain all grid points which are the centres of hexagons that intersect `W`.

`hextess` constructs a tessellation of hexagons on the window `W`. If `trim=TRUE` (the default), the tessellation is restricted to the interior of `W`, so that there will be some fragmentary hexagons near the boundary of `W`. If `trim=FALSE`, the tessellation consists of all hexagons which intersect `W`.

The points of `hexgrid(...)` are the centres of the tiles of `hextess(...)` in the same order.

In the initial position of the grid or tessellation, one of the grid points (tile centres) is placed at the `origin`, which defaults to the midpoint of the bounding rectangle of `W`. The grid can be shifted relative to this origin by specifing the `offset`.

Value

The value of `hexgrid` is a point pattern (object of class `"ppp"`).

The value of `hextess` is a tessellation (object of class `"tess"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<r.turner@auckland.ac.nz>` and Ege Rubak `<rubak@math.aau.dk>`

See Also

`tess`

`hexagon`

Examples

```
if(interactive()) {
  W <- Window(chorley)
  s <- 0.7
} else {
  W <- letterR
  s <- 0.3
}
plot(hextess(W, s))
plot(hexgrid(W, s), add=TRUE)
```

HierHard *The Hierarchical Hard Core Point Process Model*

Description

Creates an instance of the hierarchical hard core point process model which can then be fitted to point pattern data.

Usage

```
HierHard(hradii=NULL, types=NULL, archy=NULL)
```

Arguments

`hradii`	Optional matrix of hard core distances
`types`	Optional; vector of all possible types (i.e. the possible levels of the `marks` variable in the data)
`archy`	Optional: the hierarchical order. See Details.

Details

This is a hierarchical point process model for a multitype point pattern (Högmander and Särkkä, 1999; Grabarnik and Särkkä, 2009). It is appropriate for analysing multitype point pattern data in which the types are ordered so that the points of type j depend on the points of type $1, 2, \ldots, j-1$.

The hierarchical version of the (stationary) hard core process with m types, with hard core distances h_{ij} and parameters β_j, is a point process in which each point of type j contributes a factor β_j to the probability density of the point pattern. If any pair of points of types i and j lies closer than h_{ij} units apart, the configuration of points is impossible (probability density zero).

The nonstationary hierarchical hard core process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location and type, rather than a constant beta.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the hierarchical hard core process pairwise interaction is yielded by the function `HierHard()`. See the examples below.

The argument `types` need not be specified in normal use. It will be determined automatically from the point pattern data set to which the HierHard interaction is applied, when the user calls `ppm`. However, the user should be confident that the ordering of types in the dataset corresponds to the ordering of rows and columns in the matrix `radii`.

The argument `archy` can be used to specify a hierarchical ordering of the types. It can be either a vector of integers or a character vector matching the possible types. The default is the sequence $1, 2, \ldots, m$ meaning that type j depends on types $1, 2, \ldots, j-1$.

The matrix `iradii` must be square, with entries which are either positive numbers, or zero or `NA`. A value of zero or `NA` indicates that no hard core interaction term should be included for this combination of types.

Note that only the hard core distances are specified in `HierHard`. The canonical parameters $\log(\beta_j)$ are estimated by `ppm()`, not fixed in `HierHard()`.

Value

An object of class "interact" describing the interpoint interaction structure of the hierarchical hard core process with hard core distances $hradii[i, j]$.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

References

Grabarnik, P. and Särkkä, A. (2009) Modelling the spatial structure of forest stands by multivariate point processes with hierarchical interactions. *Ecological Modelling* **220**, 1232–1240.

Högmander, H. and Särkkä, A. (1999) Multitype spatial point patterns with hierarchical interactions. *Biometrics* **55**, 1051–1058.

See Also

MultiHard for the corresponding symmetrical interaction.

HierStrauss, HierStraussHard.

Examples

```
h <- matrix(c(4, NA, 10, 15), 2, 2)
HierHard(h)
# prints a sensible description of itself
ppm(ants ~1, HierHard(h))
# fit the stationary hierarchical hard core process to ants data
```

---

hierpair.family *Hierarchical Pairwise Interaction Process Family*

Description

An object describing the family of all hierarchical pairwise interaction Gibbs point processes.

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the hierarchical pairwise interaction family of point process models.

Anyway, hierpair.family is an object of class "isf" containing a function hierpair.family$eval for evaluating the sufficient statistics of any hierarchical pairwise interaction point process model taking an exponential family form.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

See Also

Other families: pairwise.family, pairsat.family, ord.family, inforder.family.

Hierarchical Strauss interaction: HierStrauss.

HierStrauss *The Hierarchical Strauss Point Process Model*

Description

Creates an instance of the hierarchical Strauss point process model which can then be fitted to point pattern data.

Usage

```
HierStrauss(radii, types=NULL, archy=NULL)
```

Arguments

radii	Matrix of interaction radii
types	Optional; vector of all possible types (i.e. the possible levels of the marks variable in the data)
archy	Optional: the hierarchical order. See Details.

Details

This is a hierarchical point process model for a multitype point pattern (Högmander and Särkkä, 1999; Grabarnik and Särkkä, 2009). It is appropriate for analysing multitype point pattern data in which the types are ordered so that the points of type j depend on the points of type $1, 2, \ldots, j-1$.

The hierarchical version of the (stationary) Strauss process with m types, with interaction radii r_{ij} and parameters β_j and γ_{ij} is a point process in which each point of type j contributes a factor β_j to the probability density of the point pattern, and a pair of points of types i and j closer than r_{ij} units apart contributes a factor γ_{ij} to the density **provided** $i \le j$.

The nonstationary hierarchical Strauss process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location and type, rather than a constant beta.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the hierarchical Strauss process pairwise interaction is yielded by the function `HierStrauss()`. See the examples below.

The argument `types` need not be specified in normal use. It will be determined automatically from the point pattern data set to which the HierStrauss interaction is applied, when the user calls `ppm`. However, the user should be confident that the ordering of types in the dataset corresponds to the ordering of rows and columns in the matrix `radii`.

The argument `archy` can be used to specify a hierarchical ordering of the types. It can be either a vector of integers or a character vector matching the possible types. The default is the sequence $1, 2, \ldots, m$ meaning that type j depends on types $1, 2, \ldots, j-1$.

The matrix `radii` must be symmetric, with entries which are either positive numbers or `NA`. A value of `NA` indicates that no interaction term should be included for this combination of types.

Note that only the interaction radii are specified in `HierStrauss`. The canonical parameters $\log(\beta_j)$ and $\log(\gamma_{ij})$ are estimated by `ppm()`, not fixed in `HierStrauss()`.

Value

An object of class "interact" describing the interpoint interaction structure of the hierarchical Strauss process with interaction radii $radii[i, j]$.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

References

Grabarnik, P. and Särkkä, A. (2009) Modelling the spatial structure of forest stands by multivariate point processes with hierarchical interactions. *Ecological Modelling* **220**, 1232–1240.

Högmander, H. and Särkkä, A. (1999) Multitype spatial point patterns with hierarchical interactions. *Biometrics* **55**, 1051–1058.

See Also

MultiStrauss for the corresponding symmetrical interaction.

HierHard, HierStraussHard.

Examples

```
r <- matrix(10 * c(3,4,4,3), nrow=2,ncol=2)
HierStrauss(r)
# prints a sensible description of itself
ppm(ants ~1, HierStrauss(r, , c("Messor", "Cataglyphis")))
# fit the stationary hierarchical Strauss process to ants data
```

HierStraussHard *The Hierarchical Strauss Hard Core Point Process Model*

Description

Creates an instance of the hierarchical Strauss-hard core point process model which can then be fitted to point pattern data.

Usage

```
HierStraussHard(iradii, hradii=NULL, types=NULL, archy=NULL)
```

Arguments

iradii	Matrix of interaction radii
hradii	Optional matrix of hard core distances
types	Optional; vector of all possible types (i.e. the possible levels of the marks variable in the data)
archy	Optional: the hierarchical order. See Details.

Details

This is a hierarchical point process model for a multitype point pattern (Högmander and Särkkä, 1999; Grabarnik and Särkkä, 2009). It is appropriate for analysing multitype point pattern data in which the types are ordered so that the points of type j depend on the points of type $1, 2, \ldots, j-1$.

The hierarchical version of the (stationary) Strauss hard core process with m types, with interaction radii r_{ij}, hard core distances h_{ij} and parameters β_j and γ_{ij} is a point process in which each point of type j contributes a factor β_j to the probability density of the point pattern, and a pair of points of types i and j closer than r_{ij} units apart contributes a factor γ_{ij} to the density **provided** $i \leq j$. If any pair of points of types i and j lies closer than h_{ij} units apart, the configuration of points is impossible (probability density zero).

The nonstationary hierarchical Strauss hard core process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location and type, rather than a constant beta.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the hierarchical Strauss hard core process pairwise interaction is yielded by the function `HierStraussHard()`. See the examples below.

The argument `types` need not be specified in normal use. It will be determined automatically from the point pattern data set to which the HierStraussHard interaction is applied, when the user calls `ppm`. However, the user should be confident that the ordering of types in the dataset corresponds to the ordering of rows and columns in the matrix `radii`.

The argument `archy` can be used to specify a hierarchical ordering of the types. It can be either a vector of integers or a character vector matching the possible types. The default is the sequence $1, 2, \ldots, m$ meaning that type j depends on types $1, 2, \ldots, j-1$.

The matrices `iradii` and `hradii` must be square, with entries which are either positive numbers or zero or `NA`. A value of zero or `NA` indicates that no interaction term should be included for this combination of types.

Note that only the interaction radii and hard core distances are specified in `HierStraussHard`. The canonical parameters $\log(\beta_j)$ and $\log(\gamma_{ij})$ are estimated by `ppm()`, not fixed in `HierStraussHard()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the hierarchical Strauss-hard core process with interaction radii $iradii[i, j]$ and hard core distances $hradii[i, j]$.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

, Rolf Turner `<r.turner@auckland.ac.nz>`

and Ege Rubak `<rubak@math.aau.dk>`.

References

Grabarnik, P. and Särkkä, A. (2009) Modelling the spatial structure of forest stands by multivariate point processes with hierarchical interactions. *Ecological Modelling* **220**, 1232–1240.

Högmander, H. and Särkkä, A. (1999) Multitype spatial point patterns with hierarchical interactions. *Biometrics* **55**, 1051–1058.

See Also

MultiStraussHard for the corresponding symmetrical interaction.

HierHard, HierStrauss.

Examples

```
r <- matrix(c(30, NA, 40, 30), nrow=2,ncol=2)
h <- matrix(c(4, NA, 10, 15), 2, 2)
HierStraussHard(r, h)
# prints a sensible description of itself
ppm(ants ~1, HierStraussHard(r, h))
# fit the stationary hierarchical Strauss-hard core process to ants data
```

---

hist.funxy *Histogram of Values of a Spatial Function*

Description

Computes and displays a histogram of the values of a spatial function of class "funxy".

Usage

```
  ## S3 method for class 'funxy'
hist(x, ..., xname)
```

Arguments

x	A pixel image (object of class "funxy").
...	Arguments passed to as.im or hist.im.
xname	Optional. Character string to be used as the name of the dataset x.

Details

This function computes and (by default) displays a histogram of the values of the function x.

An object of class "funxy" describes a function of spatial location. It is a function(x,y,..) in the R language, with additional attributes.

The function hist.funxy is a method for the generic function hist for the class "funxy".

The function is first converted to a pixel image using as.im, then hist.im is called to produce the histogram.

Any arguments in ... are passed to as.im to determine the pixel resolution, or to hist.im to determine the histogram breaks and to control or suppress plotting. Useful arguments include W for the spatial domain, eps,dimyx for pixel resolution, main for the main title.

Value

An object of class "histogram" as returned by hist.default. This object can be plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

spatialcdf for the cumulative distribution function of an image or function.

hist, hist.default.

For other statistical graphics such as Q-Q plots, use `as.im(X)[]` to extract the pixel values of image X, and apply the usual statistical graphics commands.

Examples

```
f <- funxy(function(x,y) {x^2}, unit.square())
hist(f)
```

---

hist.im *Histogram of Pixel Values in an Image*

Description

Computes and displays a histogram of the pixel values in a pixel image. The `hist` method for class `"im"`.

Usage

```
  ## S3 method for class 'im'
hist(x, ..., probability=FALSE, xname)
```

Arguments

x	A pixel image (object of class "im").
...	Arguments passed to hist.default or barplot.
probability	Logical. If TRUE, the histogram will be normalised to give probabilities or probability densities.
xname	Optional. Character string to be used as the name of the dataset x.

Details

This function computes and (by default) displays a histogram of the pixel values in the image `x`.

An object of class `"im"` describes a pixel image. See im.object) for details of this class.

The function `hist.im` is a method for the generic function hist for the class `"im"`.

Any arguments in `...` are passed to hist.default (for numeric valued images) or barplot (for factor or logical images). For example, such arguments control the axes, and may be used to suppress the plotting.

Value

For numeric-valued images, an object of class `"histogram"` as returned by hist.default. This object can be plotted.

For factor-valued or logical images, an object of class `"barplotdata"`, which can be plotted. This is a list with components called `counts` (contingency table of counts of the numbers of pixels taking each possible value), `probs` (corresponding relative frequencies) and `mids` (graphical x-coordinates of the midpoints of the bars in the barplot).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

spatialcdf for the cumulative distribution function of an image.

hist, hist.default, barplot.

For other statistical graphics such as Q-Q plots, use X[] to extract the pixel values of image X, and apply the usual statistical graphics commands.

For information about pixel images see im.object, summary.im.

Examples

```
X <- as.im(function(x,y) {x^2}, unit.square())
hist(X)
hist(cut(X,3))
```

---

hopskel *Hopkins-Skellam Test*

Description

Perform the Hopkins-Skellam test of Complete Spatial Randomness, or simply calculate the test statistic.

Usage

```
hopskel(X)

hopskel.test(X, ...,
             alternative=c("two.sided", "less", "greater",
                           "clustered", "regular"),
             method=c("asymptotic", "MonteCarlo"),
             nsim=999)
```

Arguments

X	Point pattern (object of class "ppp").
alternative	String indicating the type of alternative for the hypothesis test. Partially matched.
method	Method of performing the test. Partially matched.
nsim	Number of Monte Carlo simulations to perform, if a Monte Carlo p-value is required.
...	Ignored.

Details

Hopkins and Skellam (1954) proposed a test of Complete Spatial Randomness based on comparing nearest-neighbour distances with point-event distances.

If the point pattern `X` contains `n` points, we first compute the nearest-neighbour distances $P_1, \ldots, P_n$ so that P_i is the distance from the ith data point to the nearest other data point. Then we generate another completely random pattern `U` with the same number `n` of points, and compute for each point of `U` the distance to the nearest point of `X`, giving distances $I_1, \ldots, I_n$. The test statistic is

$$A = \frac{\sum_i P_i^2}{\sum_i I_i^2}$$

The null distribution of A is roughly an F distribution with shape parameters $(2n, 2n)$. (This is equivalent to using the test statistic $H = A/(1 + A)$ and referring H to the Beta distribution with parameters (n, n)).

The function `hopskel` calculates the Hopkins-Skellam test statistic A, and returns its numeric value. This can be used as a simple summary of spatial pattern: the value $H = 1$ is consistent with Complete Spatial Randomness, while values $H < 1$ are consistent with spatial clustering, and values $H > 1$ are consistent with spatial regularity.

The function `hopskel.test` performs the test. If `method="asymptotic"` (the default), the test statistic H is referred to the F distribution. If `method="MonteCarlo"`, a Monte Carlo test is performed using `nsim` simulated point patterns.

Value

The value of `hopskel` is a single number.

The value of `hopskel.test` is an object of class `"htest"` representing the outcome of the test. It can be printed.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Hopkins, B. and Skellam, J.G. (1954) A new method of determining the type of distribution of plant individuals. *Annals of Botany* **18**, 213–227.

See Also

`clarkevans`, `clarkevans.test`, `nndist`, `nncross`

Examples

```
hopskel(redwood)
hopskel(redwood)
hopskel.test(redwood, alternative="clustered")
```

Hybrid *Hybrid Interaction Point Process Model*

Description

Creates an instance of a hybrid point process model which can then be fitted to point pattern data.

Usage

```
Hybrid(...)
```

Arguments

`...`	Two or more interactions (objects of class `"interact"`) or objects which can be converted to interactions. See Details.

Details

A *hybrid* (Baddeley, Turner, Mateu and Bevan, 2013) is a point process model created by combining two or more point process models, or an interpoint interaction created by combining two or more interpoint interactions.

The *hybrid* of two point processes, with probability densities $f(x)$ and $g(x)$ respectively, is the point process with probability density

$$h(x) = c\, f(x)\, g(x)$$

where c is a normalising constant.

Equivalently, the hybrid of two point processes with conditional intensities $\lambda(u, x)$ and $\kappa(u, x)$ is the point process with conditional intensity

$$\phi(u, x) = \lambda(u, x)\, \kappa(u, x).$$

The hybrid of $m > 3$ point processes is defined in a similar way.

The function `ppm`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of a hybrid interaction is yielded by the function `Hybrid()`.

The arguments `...` will be interpreted as interpoint interactions (objects of class `"interact"`) and the result will be the hybrid of these interactions. Each argument must either be an interpoint interaction (object of class `"interact"`), or a point process model (object of class `"ppm"`) from which the interpoint interaction will be extracted.

The arguments `...` may also be given in the form `name=value`. This is purely cosmetic: it can be used to attach simple mnemonic names to the component interactions, and makes the printed output from `print.ppm` neater.

Value

An object of class `"interact"` describing an interpoint interaction structure.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`
and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Baddeley, A., Turner, R., Mateu, J. and Bevan, A. (2013) Hybrids of Gibbs point process models and their implementation. *Journal of Statistical Software* **55**:11, 1–43. http://www.jstatsoft.org/v55/i11/

See Also

ppm

Examples

```
Hybrid(Strauss(0.1), Geyer(0.2, 3))

Hybrid(Ha=Hardcore(0.05), St=Strauss(0.1), Ge=Geyer(0.2, 3))

fit <- ppm(redwood, ~1, Hybrid(A=Strauss(0.02), B=Geyer(0.1, 2)))
fit

ctr <- rmhcontrol(nrep=5e4, expand=1)
plot(simulate(fit, control=ctr))

# hybrid components can be models (including hybrid models)
Hybrid(fit, S=Softcore(0.5))

# plot.fii only works if every component is a pairwise interaction
data(swedishpines)
fit2 <- ppm(swedishpines, ~1, Hybrid(DG=DiggleGratton(2,10), S=Strauss(5)))
plot(fitin(fit2))
plot(fitin(fit2), separate=TRUE, mar.panel=rep(4,4))
```

---

hybrid.family *Hybrid Interaction Family*

Description

An object describing the family of all hybrid interactions.

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the family of all hybrid point process models.

If you need to create a specific hybrid interaction model for use in modelling, use the function Hybrid.

Anyway, hybrid.family is an object of class "isf" containing a function hybrid.family$eval for evaluating the sufficient statistics of any hybrid interaction point process model.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Use `Hybrid` to make hybrid interactions.

Other families: `pairwise.family`, `pairsat.family`, `ord.family`, `inorder.family`.

---

hyperframe — *Hyper Data Frame*

Description

Create a hyperframe: a two-dimensional array in which each column consists of values of the same atomic type (like the columns of a data frame) or objects of the same class.

Usage

```
hyperframe(...,
          row.names=NULL, check.rows=FALSE, check.names=TRUE,
          stringsAsFactors=default.stringsAsFactors())
```

Arguments

`...` Arguments of the form `value` or `tag=value`. Each `value` is either an atomic vector, or a list of objects of the same class, or a single atomic value, or a single object. Each `value` will become a column of the array. The `tag` determines the name of the column. See Details.

`row.names,check.rows,check.names,stringsAsFactors`
Arguments passed to `data.frame` controlling the names of the rows, whether to check that rows are consistent, whether to check validity of the column names, and whether to convert character columns to factors.

Details

A hyperframe is like a data frame, except that its entries can be objects of any kind.

A hyperframe is a two-dimensional array in which each column consists of values of one atomic type (as in a data frame) or consists of objects of one class.

The arguments `...` are any number of arguments of the form `value` or `tag=value`. Each `value` will become a column of the array. The `tag` determines the name of the column.

Each `value` can be either

- an atomic vector or factor (i.e. numeric vector, integer vector, character vector, logical vector, complex vector or factor)
- a list of objects which are all of the same class
- one atomic value, which will be replicated to make an atomic vector or factor
- one object, which will be replicated to make a list of objects.

All columns (vectors, factors and lists) must be of the same length, if their length is greater than 1.

Value

An object of class `"hyperframe"`.

Methods for Hyperframes

There are methods for print, plot, summary, with, split, [, [<,$, $<-, names, as.data.frame as.list, cbind and rbind for the class of hyperframes. There is also is.hyperframe and as.hyperframe.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

as.hyperframe, as.hyperframe.ppx, plot.hyperframe, [.hyperframe, with.hyperframe, split.hyperframe, as.data.frame.hyperframe, cbind.hyperframe, rbind.hyperframe

Examples

```
 # equivalent to a data frame
  hyperframe(X=1:10, Y=3)

 # list of functions
  hyperframe(f=list(sin, cos, tan))

 # table of functions and matching expressions
  hyperframe(f=list(sin, cos, tan),
             e=list(expression(sin(x)), expression(cos(x)), expression(tan(x))))

  hyperframe(X=1:10, Y=letters[1:10], Z=factor(letters[1:10]),
    stringsAsFactors=FALSE)

  lambda <- runif(4, min=50, max=100)
  X <- lapply(as.list(lambda), function(x) { rpoispp(x) })
  h <- hyperframe(lambda=lambda, X=X)
  h

  h$lambda2 <- lambda^2
  h[, "lambda3"] <- lambda^3
  h[, "Y"] <- X
```

---

identify.ppp *Identify Points in a Point Pattern*

Description

If a point pattern is plotted in the graphics window, this function will find the point of the pattern which is nearest to the mouse position, and print its mark value (or its serial number if there is no mark).

Usage

```
  ## S3 method for class 'ppp'
identify(x, ...)

  ## S3 method for class 'lpp'
identify(x, ...)
```

Arguments

x	A point pattern (object of class "ppp" or "lpp").
...	Arguments passed to identify.default.

Details

This is a method for the generic function identify for point pattern objects.

The point pattern x should first be plotted using plot.ppp or plot.lpp as appropriate. Then identify(x) reads the position of the graphics pointer each time the left mouse button is pressed. It then finds the point of the pattern x closest to the mouse position. If this closest point is sufficiently close to the mouse pointer, its index (and its mark if any) will be returned as part of the value of the call.

Each time a point of the pattern is identified, text will be displayed next to the point, showing its serial number (if x is unmarked) or its mark value (if x is marked).

Value

If x is unmarked, the result is a vector containing the serial numbers of the points in the pattern x that were identified. If x is marked, the result is a 2-column matrix, the first column containing the serial numbers and the second containing the marks for these points.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

identify, clickppp

---

identify.psp — *Identify Segments in a Line Segment Pattern*

Description

If a line segment pattern is plotted in the graphics window, this function will find the segment which is nearest to the mouse position, and print its serial number.

Usage

```
  ## S3 method for class 'psp'
identify(x, ..., labels=seq_len(nsegments(x)), n=nsegments(x), plot=TRUE)
```

Arguments

x	A line segment pattern (object of class "psp").
labels	Labels associated with the segments, to be plotted when the segments are identified. A character vector or numeric vector of length equal to the number of segments in x.
n	Maximum number of segments to be identified.
plot	Logical. Whether to plot the labels when a segment is identified.
...	Arguments passed to text.default controlling the plotting of the labels.

Details

This is a method for the generic function `identify` for line segment pattern objects.

The line segment pattern `x` should first be plotted using `plot.psp`. Then `identify(x)` reads the position of the graphics pointer each time the left mouse button is pressed. It then finds the segment in the pattern `x` that is closest to the mouse position. This segment's index will be returned as part of the value of the call.

Each time a segment is identified, text will be displayed next to the point, showing its serial number (or the relevant entry of `labels`).

Value

Vector containing the serial numbers of the segments in the pattern `x` that were identified.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`identify`, `identify.ppp`.

`idw` *Inverse-distance weighted smoothing of observations at irregular points*

Description

Performs spatial smoothing of numeric values observed at a set of irregular locations using inverse-distance weighting.

Usage

```
idw(X, power=2, at="pixels", ...)
```

Arguments

`X`	A marked point pattern (object of class "ppp").
`power`	Numeric. Power of distance used in the weighting.
`at`	String specifying whether to compute the intensity values at a grid of pixel locations (`at="pixels"`) or only at the points of `X` (`at="points"`).
`...`	Arguments passed to `as.mask` to control the pixel resolution of the result.

Details

This function performs spatial smoothing of numeric values observed at a set of irregular locations.

Smoothing is performed by inverse distance weighting. If the observed values are $v_1, \ldots, v_n$ at locations $x_1, \ldots, x_n$ respectively, then the smoothed value at a location u is

$$g(u) = \frac{\sum_i w_i v_i}{\sum_i w_i}$$

where the weights are the inverse p-th powers of distance,

$$w_i = \frac{1}{d(u, x_i)^p}$$

where $d(u, x_i) = ||u - x_i||$ is the Euclidean distance from u to x_i.

The argument `X` must be a marked point pattern (object of class `"ppp"`, see `ppp.object`). The points of the pattern are taken to be the observation locations x_i, and the marks of the pattern are taken to be the numeric values v_i observed at these locations.

The marks are allowed to be a data frame. Then the smoothing procedure is applied to each column of marks.

If `at="pixels"` (the default), the smoothed mark value is calculated at a grid of pixels, and the result is a pixel image. The arguments `...` control the pixel resolution. See `as.mask`.

If `at="points"`, the smoothed mark values are calculated at the data points only, using a leave-one-out rule (the mark value at a data point is excluded when calculating the smoothed value for that point).

An alternative to inverse-distance weighting is kernel smoothing, which is performed by `Smooth.ppp`.

Value

If `X` has a single column of marks:

- If `at="pixels"` (the default), the result is a pixel image (object of class `"im"`). Pixel values are values of the interpolated function.
- If `at="points"`, the result is a numeric vector of length equal to the number of points in `X`. Entries are values of the interpolated function at the points of `X`.

If `X` has a data frame of marks:

- If `at="pixels"` (the default), the result is a named list of pixel images (object of class `"im"`). There is one image for each column of marks. This list also belongs to the class `"solist"`, for which there is a plot method.
- If `at="points"`, the result is a data frame with one row for each point of `X`, and one column for each column of marks. Entries are values of the interpolated function at the points of `X`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<r.turner@auckland.ac.nz>` and Ege Rubak `<rubak@math.aau.dk>`.

See Also

`density.ppp`, `ppp.object`, `im.object`.

See `Smooth.ppp` for kernel smoothing and `nnmark` for nearest-neighbour interpolation.

To perform other kinds of interpolation, see also the `akima` package.

Examples

```
# data frame of marks: trees marked by diameter and height
data(finpines)
plot(idw(finpines))
idw(finpines, at="points")[1:5,]
```

Iest *Estimate the I-function*

Description

Estimates the summary function $I(r)$ for a multitype point pattern.

Usage

```
Iest(X, ..., eps=NULL, r=NULL, breaks=NULL, correction=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of $I(r)$ will be computed. An object of class "ppp", or data in any format acceptable to `as.ppp()`.
...	Ignored.
eps	the resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
r	Optional. Numeric vector of values for the argument r at which $I(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on `r`.
breaks	This argument is for internal use only.
correction	Optional. Vector of character strings specifying the edge correction(s) to be used by `Jest`.

Details

The I function summarises the dependence between types in a multitype point process (Van Lieshout and Baddeley, 1999) It is based on the concept of the J function for an unmarked point process (Van Lieshout and Baddeley, 1996). See `Jest` for information about the J function.

The I function is defined as

$$I(r) = \sum_{i=1}^{m} p_i J_{ii}(r) - J_{\bullet\bullet}(r)$$

where $J_{\bullet\bullet}$ is the J function for the entire point process ignoring the marks, while J_{ii} is the J function for the process consisting of points of type i only, and p_i is the proportion of points which are of type i.

The I function is designed to measure dependence between points of different types, even if the points are not Poisson. Let X be a stationary multitype point process, and write X_i for the process of points of type i. If the processes X_i are independent of each other, then the I-function is identically equal to 0. Deviations $I(r) < 1$ or $I(r) > 1$ typically indicate negative and positive association, respectively, between types. See Van Lieshout and Baddeley (1999) for further information.

An estimate of I derived from a multitype spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern. The estimate of $I(r)$ is compared against the constant function 0. Deviations $I(r) < 1$ or $I(r) > 1$ may suggest negative and positive association, respectively.

This algorithm estimates the I-function from the multitype point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial marked point process in the plane, observed through a bounded window.

The argument `X` is interpreted as a point pattern object (of class `"ppp"`, see `ppp.object`) and can be supplied in any of the formats recognised by `as.ppp()`. It must be a multitype point pattern (it must have a `marks` vector which is a `factor`).

The function `Jest` is called to compute estimates of the J functions in the formula above. In fact three different estimates are computed using different edge corrections. See `Jest` for information.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing

`r`	the vector of values of the argument r at which the function I has been estimated
`rs`	the "reduced sample" or "border correction" estimator of $I(r)$ computed from the border-corrected estimates of J functions
`km`	the spatial Kaplan-Meier estimator of $I(r)$ computed from the Kaplan-Meier estimates of J functions
`han`	the Hanisch-style estimator of $I(r)$ computed from the Hanisch-style estimates of J functions
`un`	the uncorrected estimate of $I(r)$ computed from the uncorrected estimates of J
`theo`	the theoretical value of $I(r)$ for a stationary Poisson process: identically equal to 0

Note

Sizeable amounts of memory may be needed during the calculation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

`Jest`

Examples

```
data(amacrine)
Ic <- Iest(amacrine)
plot(Ic, main="Amacrine Cells data")
# values are below I= 0, suggesting negative association
# between 'on' and 'off' cells.
```

`im` *Create a Pixel Image Object*

Description

Creates an object of class `"im"` representing a two-dimensional pixel image.

Usage

```
im(mat, xcol=seq_len(ncol(mat)), yrow=seq_len(nrow(mat)),
 xrange=NULL, yrange=NULL,
 unitname=NULL)
```

Arguments

`mat`	matrix or vector containing the pixel values of the image.
`xcol`	vector of x coordinates for the pixel grid
`yrow`	vector of y coordinates for the pixel grid
`xrange,yrange`	Optional. Vectors of length 2 giving the x and y limits of the enclosing rectangle. (Ignored if `xcol`, `yrow` are present.)
`unitname`	Optional. Name of unit of length. Either a single character string, or a vector of two character strings giving the singular and plural forms, respectively.

Details

This function creates an object of class `"im"` representing a 'pixel image' or two-dimensional array of values.

The pixel grid is rectangular and occupies a rectangular window in the spatial coordinate system. The pixel values are *scalars*: they can be real numbers, integers, complex numbers, single characters or strings, logical values, or categorical values. A pixel's value can also be `NA`, meaning that no value is defined at that location, and effectively that pixel is 'outside' the window. Although the pixel values must be scalar, photographic colour images (i.e., with red, green, and blue brightness channels) can be represented as character-valued images in **spatstat**, using R's standard encoding of colours as character strings.

The matrix `mat` contains the 'greyscale' values for a rectangular grid of pixels. Note carefully that the entry `mat[i,j]` gives the pixel value at the location `(xcol[j],yrow[i])`. That is, the **row** index of the matrix `mat` corresponds to increasing **y** coordinate, while the column index of `mat` corresponds to increasing **x** coordinate. Thus `yrow` has one entry for each row of `mat` and `xcol` has one entry for each column of `mat`. Under the usual convention in R, a correct display of the image would be obtained by transposing the matrix, e.g. `image.default(xcol, yrow, t(mat))`, if you wanted to do it by hand.

The entries of mat may be numeric (real or integer), complex, logical, character, or factor values. If mat is not a matrix, it will be converted into a matrix with nrow(mat) = length(yrow) and ncol(mat) = length(xcol).

To make a factor-valued image, note that R has a quirky way of handling matrices with factor-valued entries. The command matrix cannot be used directly, because it destroys factor information. To make a factor-valued image, do one of the following:

- Create a factor containing the pixel values, say mat <- factor(.....), and then assign matrix dimensions to it by dim(mat) <- c(nr, nc) where nr, nc are the numbers of rows and columns. The resulting object mat is both a factor and a vector.
- Supply mat as a one-dimensional factor and specify the arguments xcol and yrow to determine the dimensions of the image.
- Use the functions cut.im or eval.im to make factor-valued images from other images).

For a description of the methods available for pixel image objects, see im.object.

To convert other kinds of data to a pixel image (for example, functions or windows), use as.im.

Warnings

The internal representation of images is likely to change in future releases of **spatstat**. The safe way to extract pixel values from an image object is to use as.matrix.im or [.im.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

im.object for details of the class.

as.im for converting other kinds of data to an image.

as.matrix.im, [.im, eval.im for manipulating images.

Examples

```
vec <- rnorm(1200)
mat <- matrix(vec, nrow=30, ncol=40)
whitenoise <- im(mat)
whitenoise <- im(mat, xrange=c(0,1), yrange=c(0,1))
whitenoise <- im(mat, xcol=seq(0,1,length=40), yrow=seq(0,1,length=30))
whitenoise <- im(vec, xcol=seq(0,1,length=40), yrow=seq(0,1,length=30))
plot(whitenoise)

# Factor-valued images:
f <- factor(letters[1:12])
dim(f) <- c(3,4)
Z <- im(f)

# Factor image from other image:
cutwhite <- cut(whitenoise, 3)
plot(cutwhite)

# Factor image from raw data
```

```
cutmat <- cut(mat, 3)
dim(cutmat) <- c(30,40)
cutwhite <- im(cutmat)
plot(cutwhite)
```

---

im.apply *Apply Function Pixelwise to List of Images*

Description

Returns a pixel image obtained by applying a function to the values of corresponding pixels in several pixel images.

Usage

```
im.apply(X, FUN, ...)
```

Arguments

X	A list of pixel images (objects of class "im").
FUN	A function that can be applied to vectors, or a character string giving the name of such a function.
...	Additional arguments to FUN.

Details

The argument X should be a list of pixel images (objects of class "im"). If the images do not have identical pixel grids, they will be converted to a common grid using harmonise.im.

At each pixel location, the values of the images in X at that pixel will be extracted as a vector. The function FUN will be applied to this vector. The result (which should be a single value) becomes the pixel value of the resulting image.

Value

A pixel image (object of class "im").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

eval.im for algebraic operations with images.

Examples

```
DA <- density(split(amacrine))
DA
im.apply(DA, max)
```

im.object *Class of Images*

Description

A class `"im"` to represent a two-dimensional pixel image.

Details

An object of this class represents a two-dimensional pixel image. It specifies

- the dimensions of the rectangular array of pixels
- x and y coordinates for the pixels
- a numeric value ("grey value") at each pixel

If `X` is an object of type `im`, it contains the following elements:

`v`	matrix of values
`dim`	dimensions of matrix `v`
`xrange`	range of x coordinates of image window
`yrange`	range of y coordinates of image window
`xstep`	width of one pixel
`ystep`	height of one pixel
`xcol`	vector of x coordinates of centres of pixels
`yrow`	vector of y coordinates of centres of pixels

Users are strongly advised not to manipulate these entries directly.

Objects of class `"im"` may be created by the functions `im` and `as.im`. Image objects are also returned by various functions including `distmap`, `Kmeasure`, `setcov`, `eval.im` and `cut.im`.

Image objects may be displayed using the methods `plot.im`, `image.im`, `persp.im` and `contour.im`. There are also methods `print.im` for printing information about an image, `summary.im` for summarising an image, `mean.im` for calculating the average pixel value, `hist.im` for plotting a histogram of pixel values, `quantile.im` for calculating quantiles of pixel values, and `cut.im` for dividing the range of pixel values into categories.

Pixel values in an image may be extracted using the subset operator `[.im`. To extract all pixel values from an image object, use `as.matrix.im`. The levels of a factor-valued image can be extracted and changed with `levels` and `levels<-`.

Calculations involving one or more images (for example, squaring all the pixel values in an image, converting numbers to factor levels, or subtracting one image from another) can often be done easily using `eval.im`. To find all pixels satisfying a certain constraint, use `solutionset`.

Note carefully that the entry `v[i,j]` gives the pixel value at the location `(xcol[j],yrow[i]`. That is, the **row** index of the matrix `v` corresponds to increasing **y** coordinate, while the column index of `mat` corresponds to increasing **x** coordinate. Thus `yrow` has one entry for each row of `v` and `xcol` has one entry for each column of `v`. Under the usual convention in R, a correct display of the image would be obtained by transposing the matrix, e.g. `image.default(xcol, yrow, t(v))`, if you wanted to do it by hand.

Warnings

The internal representation of images is likely to change in future releases of **spatstat**. Do not address the entries in an image directly. To extract all pixel values from an image object, use as.matrix.im.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

im, as.im, plot.im, persp.im, eval.im, [.im

---

imcov *Spatial Covariance of a Pixel Image*

Description

Computes the unnormalised spatial covariance function of a pixel image.

Usage

```
imcov(X, Y=X)
```

Arguments

X	A pixel image (object of class "im".
Y	Optional. Another pixel image.

Details

The (uncentred, unnormalised) *spatial covariance function* of a pixel image X in the plane is the function $C(v)$ defined for each vector v as

$$C(v) = \int X(u)X(u-v)\,\mathrm{d}u$$

where the integral is over all spatial locations u, and where $X(u)$ denotes the pixel value at location u.

This command computes a discretised approximation to the spatial covariance function, using the Fast Fourier Transform. The return value is another pixel image (object of class "im") whose greyscale values are values of the spatial covariance function.

If the argument Y is present, then imcov(X,Y) computes the set *cross-covariance* function $C(u)$ defined as

$$C(v) = \int X(u)Y(u-v)\,\mathrm{d}u.$$

Note that imcov(X,Y) is equivalent to convolve.im(X,Y,reflectY=TRUE).

Value

A pixel image (an object of class "im") representing the spatial covariance function of X, or the cross-covariance of X and Y.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

setcov, convolve.im, owin, as.owin, erosion

Examples

```
X <- as.im(square(1))
v <- imcov(X)
plot(v)
```

---

improve.kppm *Improve Intensity Estimate of Fitted Cluster Point Process Model*

Description

Update the fitted intensity of a fitted cluster point process model.

Usage

```
improve.kppm(object, type=c("quasi", "wclik1", "clik1"), rmax = NULL,
             eps.rmax = 0.01, dimyx = 50, maxIter = 100, tolerance = 1e-06,
             fast = TRUE, vcov = FALSE, fast.vcov = FALSE, verbose = FALSE,
                           save.internals = FALSE)
```

Arguments

object	Fitted cluster point process model (object of class "kppm").
type	A character string indicating the method of estimation. Current options are "clik1", "wclik1" and "quasi" for, respectively, first order composite (Poisson) likelihood, weighted first order composite likelihood and quasi-likelihood.
rmax	Optional. The dependence range. Not usually specified by the user.
eps.rmax	Numeric. A small positive number which is used to determine rmax from the tail behaviour of the pair correlation function. Namely rmax is the smallest value of r at which $(g(r) - 1)/(g(0) - 1)$ falls below eps.rmax. Ignored if rmax is provided.
dimyx	Pixel array dimensions. See Details.
maxIter	Integer. Maximum number of iterations of iterative weighted least squares (Fisher scoring).
tolerance	Numeric. Tolerance value specifying when to stop iterative weighted least squares (Fisher scoring).

`fast`	Logical value indicating whether tapering should be used to make the computations faster (requires the package **Matrix**).
`vcov`	Logical value indicating whether to calculate the asymptotic variance covariance/matrix.
`fast.vcov`	Logical value indicating whether tapering should be used for the variance/covariance matrix to make the computations faster (requires the package **Matrix**). Caution: This is expected to underestimate the true asymptotic variances/covariances.
`verbose`	A logical indicating whether the details of computations should be printed.
`save.internals`	A logical indicating whether internal quantities should be saved in the returned object (mostly for development purposes).

Details

This function reestimates the intensity parameters in a fitted `"kppm"` object. If `type="clik1"` estimates are based on the first order composite (Poisson) likelihood, which ignores dependence between the points. Note that `type="clik1"` is mainly included for testing purposes and is not recommended for the typical user; instead the more efficient `kppm` with `improve.type="none"` should be used.

When `type="quasi"` or `type="wclik1"` the dependence structure between the points is incorporated in the estimation procedure by using the estimated pair correlation function in the estimating equation.

In all cases the estimating equation is based on dividing the observation window into small subregions and count the number of points in each subregion. To do this the observation window is first converted into a digital mask by `as.mask` where the resolution is controlled by the argument `dimyx`. The computational time grows with the cube of the number of subregions, so fine grids may take very long to compute (or even run out of memory).

Value

A fitted cluster point process model of class `"kppm"`.

Author(s)

Abdollah Jalilian <jalilian@razi.ac.ir>

and Rasmus Waagepetersen <rw@math.aau.dk> adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.e

and Ege Rubak <rubak@math.aau.dk>

References

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes, *Biometrics*, **63**, 252-258.

Guan, Y. and Shen, Y. (2010) A weighted estimating equation approach to inference for inhomogeneous spatial point processes, *Biometrika*, **97**, 867-880.

Guan, Y., Jalilian, A. and Waagepetersen, R. (2015) Quasi-likelihood for spatial point processes. *Journal of the Royal Statistical Society, Series B* **77**, 677–697.

See Also

`ppm`, `kppm`, `improve.kppm`

Examples

```
# fit a Thomas process using minimum contrast estimation method
# to model interaction between points of the pattern
fit0 <- kppm(bei ~ elev + grad, data = bei.extra)

# fit the log-linear intensity model with quasi-likelihood method
fit1 <- improve.kppm(fit0, type="quasi")

# compare
coef(fit0)
coef(fit1)
```

---

incircle *Find Largest Circle Inside Window*

Description

Find the largest circle contained in a given window.

Usage

```
incircle(W)

inradius(W)
```

Arguments

W	A window (object of class "owin").

Details

Given a window W of any type and shape, the function incircle determines the largest circle that is contained inside W, while inradius computes its radius only.

For non-rectangular windows, the incircle is computed approximately by finding the maximum of the distance map (see distmap) of the complement of the window.

Value

The result of incircle is a list with entries x,y,r giving the location (x,y) and radius r of the incircle.

The result of inradius is the numerical value of radius.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

centroid.owin

Examples

```
W <- square(1)
Wc <- incircle(W)
plot(W)
plot(disc(Wc$r, c(Wc$x, Wc$y)), add=TRUE)

plot(letterR)
Rc <- incircle(letterR)
plot(disc(Rc$r, c(Rc$x, Rc$y)), add=TRUE)

W <- as.mask(letterR)
plot(W)
Rc <- incircle(W)
plot(disc(Rc$r, c(Rc$x, Rc$y)), add=TRUE)
```

`increment.fv` *Increments of a Function*

Description

Compute the change in the value of a function `f` when the function argument increases by `delta`.

Usage

```
increment.fv(f, delta)
```

Arguments

`f`	Object of class `"fv"` representing a function.
`delta`	Numeric. The increase in the value of the function argument.

Details

This command computes the new function

$$g(x) = f(x + h) - f(x - h)$$

where `h = delta/2`. The value of $g(x)$ is the change in the value of f over an interval of length `delta` centred at x.

Value

Another object of class `"fv"` compatible with `X`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`fv.object`, `deriv.fv`

Examples

```
plot(increment.fv(Kest(cells), 0.05))
```

`infline` *Infinite Straight Lines*

Description

Define the coordinates of one or more straight lines in the plane

Usage

```
infline(a = NULL, b = NULL, h = NULL, v = NULL, p = NULL, theta = NULL)

## S3 method for class 'infline'
print(x, ...)

## S3 method for class 'infline'
plot(x, ...)
```

Arguments

`a,b`	Numeric vectors of equal length giving the intercepts a and slopes b of the lines. Incompatible with `h,v,p,theta`
`h`	Numeric vector giving the positions of horizontal lines when they cross the y axis. Incompatible with `a,b,v,p,theta`
`v`	Numeric vector giving the positions of vertical lines when they cross the x axis. Incompatible with `a,b,h,p,theta`
`p,theta`	Numeric vectors of equal length giving the polar coordinates of the line. Incompatible with `a,b,h,v`
`x`	An object of class `"infline"`
`...`	Extra arguments passed to `print` for printing or `abline` for plotting

Details

The class `infline` is a convenient way to handle infinite straight lines in the plane.

The position of a line can be specified in several ways:

- its intercept a and slope b in the equation $y = a + bx$ can be used unless the line is vertical.
- for vertical lines we can use the position v where the line crosses the y axis
- for horizontal lines we can use the position h where the line crosses the x axis
- the polar coordinates p and θ can be used for any line. The line equation is

$$y \cos\theta + x \sin\theta = p$$

The command `infline` will accept line coordinates in any of these formats. The arguments `a,b,h,v` have the same interpretation as they do in the line-plotting function `abline`.

The command `infline` converts between different coordinate systems (e.g. from `a,b` to `p,theta`) and returns an object of class `"infline"` that contains a representation of the lines in each appropriate coordinate system. This object can be printed and plotted.

Value

The value of `infline` is an object of class `"infline"` which is basically a data frame with columns `a,b,h,v,p,theta`. Each row of the data frame represents one line. Entries may be `NA` if a coordinate is not applicable to a particular line.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>` and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`rotate.infline`, `clip.infline`, `chop.tess`, `whichhalfplane`

Examples

```
infline(a=10:13,b=1)
infline(p=1:3, theta=pi/4)
plot(c(-1,1),c(-1,1),type="n",xlab="",ylab="", asp=1)
plot(infline(p=0.4, theta=seq(0,pi,length=20)))
```

`influence.ppm` *Influence Measure for Spatial Point Process Model*

Description

Computes the influence measure for a fitted spatial point process model.

Usage

```
## S3 method for class 'ppm'
influence(model, ..., drop = FALSE, iScore=NULL, iHessian=NULL, iArgs=NULL)
```

Arguments

`model`	Fitted point process model (object of class `"ppm"`).
`...`	Ignored.
`drop`	Logical. Whether to include (`drop=FALSE`) or exclude (`drop=TRUE`) contributions from quadrature points that were not used to fit the model.
`iScore,iHessian`	Components of the score vector and Hessian matrix for the irregular parameters, if required. See Details.
`iArgs`	List of extra arguments for the functions `iScore`, `iHessian` if required.

Details

Given a fitted spatial point process model model, this function computes the influence measure described in Baddeley, Chang and Song (2013).

The function influence is generic, and influence.ppm is the method for objects of class "ppm" representing point process models.

The influence of a point process model is a value attached to each data point (i.e. each point of the point pattern to which the model was fitted). The influence value $s(x_i)$ at a data point x_i represents the change in the maximised log (pseudo)likelihood that occurs when the point x_i is deleted. A relatively large value of $s(x_i)$ indicates a data point with a large influence on the fitted model.

If the point process model trend has irregular parameters that were fitted (using ippm) then the influence calculation requires the first and second derivatives of the log trend with respect to the irregular parameters. The argument iScore should be a list, with one entry for each irregular parameter, of R functions that compute the partial derivatives of the log trend (i.e. log intensity or log conditional intensity) with respect to each irregular parameter. The argument iHessian should be a list, with p^2 entries where p is the number of irregular parameters, of R functions that compute the second order partial derivatives of the log trend with respect to each pair of irregular parameters.

The result of influence.ppm is an object of class "influence.ppm". It can be plotted (by plot.influence.ppm), or converted to a marked point pattern by as.ppp (see as.ppp.influence.ppm).

Value

An object of class "influence.ppm" that can be plotted (by plot.influence.ppm). There are also methods for print, [, as.ppp and as.owin.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A. and Chang, Y.M. and Song, Y. (2013) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics* **40**, 86–104.

See Also

leverage.ppm, dfbetas.ppm, ppmInfluence, plot.influence.ppm

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X ~x+y)
plot(influence(fit))
```

`inforder.family` *Infinite Order Interaction Family*

Description

An object describing the family of all Gibbs point processes with infinite interaction order.

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the interaction structure of Gibbs point processes which have infinite order of interaction, such as the area-interaction process *AreaInter*.

Anyway, `inforder.family` is an object of class `"isf"` containing a function `inforder.family$eval` for evaluating the sufficient statistics of a Gibbs point process model taking an exponential family form.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

See Also

`AreaInter` to create the area interaction process structure.

Other families: `pairwise.family`, `pairsat.family`, `ord.family`.

`insertVertices` *Insert New Vertices in a Linear Network*

Description

Adds new vertices to a linear network at specified locations along the network.

Usage

```
insertVertices(L, ...)
```

Arguments

`L`	Linear network (object of class `"linnet"`) or point pattern on a linear network (object of class `"lpp"`).
`...`	Additional arguments passed to `as.lpp` specifying the positions of the new vertices along the network.

Details

This function adds new vertices at locations along an existing linear network.

The argument `L` can be either a linear network (class `"linnet"`) or some other object that includes a linear network.

The new vertex locations can be specified either as a point pattern (class `"lpp"` or `"ppp"`) or using coordinate vectors `x,y` or `seg,tp` or `x,y,seg,tp` as explained in the help for `as.lpp`.

This function breaks the existing line segments of `L` into pieces at the locations specified by the coordinates `seg,tp` and creates new vertices at these locations.

The result is the modified object, with an attribute `"id"` such that the `i`th added vertex has become the `id[i]`th vertex of the new network.

Value

An object of the same class as `L` representing the result of adding the new vertices. The result also has an attribute `"id"` as described in Details.

Author(s)

Adrian Baddeley

See Also

`as.lpp`

Examples

```
opa <- par(mfrow=c(1,3), mar=rep(0,4))
simplenet

plot(simplenet, main="")
plot(vertices(simplenet), add=TRUE)

# add two new vertices at specified local coordinates
L <- insertVertices(simplenet, seg=c(3,7), tp=c(0.2, 0.5))
L
plot(L, main="")
plot(vertices(L), add=TRUE)
id <- attr(L, "id")
id
plot(vertices(L)[id], add=TRUE, pch=16)

# add new vertices at three randomly-generated points
X <- runiflpp(3, simplenet)
LL <- insertVertices(simplenet, X)
plot(LL, main="")
plot(vertices(LL), add=TRUE)
ii <- attr(LL, "id")
plot(vertices(LL)[ii], add=TRUE, pch=16)
par(opa)
```

inside.boxx *Test Whether Points Are Inside A Multidimensional Box*

Description

Test whether points lie inside or outside a given multidimensional box.

Usage

```
inside.boxx(..., w)
```

Arguments

...	Coordinates of points to be tested. One vector for each dimension (all of same length). (Alternatively, a single point pattern object of class "ppx" or its coordinates as a "hyperframe")
w	A window. This should be an object of class boxx, or can be given in any format acceptable to as.boxx().

Details

This function tests whether each of the points (x[i],y[i]) lies inside or outside the window w and returns TRUE if it is inside.

The boundary of the window is treated as being inside.

Normally each argument provided (except w) must be numeric vectors of equal length (length zero is allowed) containing the coordinates of points. Alternatively a single point pattern (object of class "ppx") can be given; then the coordinates of the point pattern are extracted.

Value

Logical vector whose ith entry is TRUE if the corresponding point is inside w.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

boxx, as.boxx

Examples

```
# Random points in box with side [0,2]
w <- boxx(c(0,2), c(0,2), c(0,2))

# Random points in box with side [-1,3]
x <- runif(30, min=-1, max=3)
y <- runif(30, min=-1, max=3)
z <- runif(30, min=-1, max=3)
```

```
# Points falling in smaller box
ok <- inside.boxx(x, y, z, w=w)

# Same using a point pattern as argument:
X <- ppx(data = cbind(x, y, z), domain = boxx(c(0,3), c(0,3), c(0,3)))
ok2 <- inside.boxx(X, w=w)
```

---

inside.owin *Test Whether Points Are Inside A Window*

Description

Test whether points lie inside or outside a given window.

Usage

```
inside.owin(x, y, w)
```

Arguments

x	Vector of x coordinates of points to be tested. (Alternatively, a point pattern object providing both x and y coordinates.)
y	Vector of y coordinates of points to be tested.
w	A window. This should be an object of class `owin`, or can be given in any format acceptable to `as.owin()`.

Details

This function tests whether each of the points `(x[i],y[i])` lies inside or outside the window `w` and returns `TRUE` if it is inside.

The boundary of the window is treated as being inside.

If `w` is of type `"rectangle"` or `"polygonal"`, the algorithm uses analytic geometry (the discrete Stokes theorem). Computation time is linear in the number of points and (for polygonal windows) in the number of vertices of the boundary polygon. Boundary cases are correct to single precision accuracy.

If `w` is of type `"mask"` then the pixel closest to `(x[i],y[i])` is tested. The results may be incorrect for points lying within one pixel diameter of the window boundary.

Normally `x` and `y` must be numeric vectors of equal length (length zero is allowed) containing the coordinates of points. Alternatively `x` can be a point pattern (object of class `"ppp"`) while `y` is missing; then the coordinates of the point pattern are extracted.

Value

Logical vector whose `i`th entry is `TRUE` if the corresponding point `(x[i],y[i])` is inside `w`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

owin.object, as.owin

Examples

```
  # hexagonal window
  k <- 6
  theta <- 2 * pi * (0:(k-1))/k
  co <- cos(theta)
  si <- sin(theta)
  mas <- owin(c(-1,1), c(-1,1), poly=list(x=co, y=si))
  ## Not run:
  plot(mas)

## End(Not run)

  # random points in rectangle
  x <- runif(30,min=-1, max=1)
  y <- runif(30,min=-1, max=1)

  ok <- inside.owin(x, y, mas)

  ## Not run:
  points(x[ok], y[ok])
  points(x[!ok], y[!ok], pch="x")

## End(Not run)
```

---

integral.im *Integral of a Pixel Image*

Description

Computes the integral of a pixel image.

Usage

```
integral(f, domain=NULL, ...)

## S3 method for class 'im'
integral(f, domain=NULL, ...)
```

Arguments

f	A pixel image (object of class "im") with pixel values that can be treated as numeric or complex values.
domain	Optional. Window specifying the domain of integration. Alternatively a tessellation.
...	Ignored.

Details

The function integral is generic, with methods for "im", "msr", "linim" and "linfun".

The method integral.im treats the pixel image f as a function of the spatial coordinates, and computes its integral. The integral is calculated by summing the pixel values and multiplying by the area of one pixel.

The pixel values of f may be numeric, integer, logical or complex. They cannot be factor or character values.

The logical values TRUE and FALSE are converted to 1 and 0 respectively, so that the integral of a logical image is the total area of the TRUE pixels, in the same units as unitname(x).

If domain is a window (class "owin") then the integration will be restricted to this window. If domain is a tessellation (class "tess") then the integral of f in each tile of domain will be computed.

Value

A single numeric or complex value (or a vector of such values if domain is a tessellation).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

eval.im, [.im

Examples

```
# approximate integral of f(x,y) dx dy
f <- function(x,y){3*x^2 + 2*y}
Z <- as.im(f, square(1))
integral.im(Z)
# correct answer is 2

D <- density(cells)
integral.im(D)
# should be approximately equal to number of points = 42

# integrate over the subset [0.1,0.9] x [0.2,0.8]
W <- owin(c(0.1,0.9), c(0.2,0.8))
integral.im(D, W)
```

---

integral.linim *Integral on a Linear Network*

Description

Computes the integral (total value) of a function or pixel image over a linear network.

Usage

```
## S3 method for class 'linim'
integral(f, domain=NULL, ...)

## S3 method for class 'linfun'
integral(f, domain=NULL, ..., delta)
```

Arguments

`f`	A pixel image on a linear network (class "linim") or a function on a linear network (class "linfun").
`domain`	Optional window specifying the domain of integration.
`...`	Ignored.
`delta`	Optional. The step length (in coordinate units) for computing the approximate integral. A single positive number.

Details

The integral (total value of the function over the network) is calculated.

Value

A numeric value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

linim, integral.im

Examples

```
# make some data
xcoord <- linfun(function(x,y,seg,tp) { x }, simplenet)
integral(xcoord)
X <- as.linim(xcoord)
integral(X)
```

---

integral.msr *Integral of a Measure*

Description

Computes the integral (total value) of a measure over its domain.

Usage

```
## S3 method for class 'msr'
integral(f, domain=NULL, ...)
```

Arguments

f	A signed measure or vector-valued measure (object of class "msr").
domain	Optional window specifying the domain of integration. Alternatively a tessellation.
...	Ignored.

Details

The integral (total value of the measure over its domain) is calculated.

If domain is a window (class "owin") then the integration will be restricted to this window. If domain is a tessellation (class "tess") then the integral of f in each tile of domain will be computed.

For a multitype measure m, use split.msr to separate the contributions for each type of point, as shown in the Examples.

Value

A numeric value (for a signed measure) or a vector of values (for a vector-valued measure).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

msr, integral

Examples

```
fit <- ppm(cells ~ x)
rr <- residuals(fit)
integral(rr)

# vector-valued measure
rs <- residuals(fit, type="score")
integral(rs)

# multitype
fitA <- ppm(amacrine ~ x)
rrA <- residuals(fitA)
sapply(split(rrA), integral)

# multitype and vector-valued
rsA <- residuals(fitA, type="score")
sapply(split(rsA), integral)
```

`intensity` *Intensity of a Dataset or a Model*

Description

Generic function for computing the intensity of a spatial dataset or spatial point process model.

Usage

```
intensity(X, ...)
```

Arguments

`X`	A spatial dataset or a spatial point process model.
`...`	Further arguments depending on the class of `X`.

Details

This is a generic function for computing the intensity of a spatial dataset or spatial point process model. There are methods for point patterns (objects of class `"ppp"`) and fitted point process models (objects of class `"ppm"`).

The empirical intensity of a dataset is the average density (the average amount of 'stuff' per unit area or volume). The empirical intensity of a point pattern is computed by the method `intensity.ppp`.

The theoretical intensity of a stochastic model is the expected density (expected amount of 'stuff' per unit area or volume). The theoretical intensity of a fitted point process model is computed by the method `intensity.ppm`.

Value

Usually a numeric value or vector.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`intensity.ppp`, `intensity.ppm`.

`intensity.dppm` *Intensity of Determinantal Point Process Model*

Description

Extracts the intensity of a determinantal point process model.

Usage

```
  ## S3 method for class 'detpointprocfamily'
intensity(X, ...)

  ## S3 method for class 'dppm'
intensity(X, ...)
```

Arguments

`X`	A determinantal point process model (object of class `"detpointprocfamily"` or `"dppm"`).
`...`	Ignored.

Value

A numeric value (if the model is stationary), a pixel image (if the model is non-stationary) or `NA` if the intensity is unknown for the model.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

`intensity.lpp` *Empirical Intensity of Point Pattern on Linear Network*

Description

Computes the average number of points per unit length in a point pattern on a linear network.

Usage

```
## S3 method for class 'lpp'
intensity(X, ...)
```

Arguments

`X`	A point pattern on a linear network (object of class `"lpp"`).
`...`	Ignored.

Details

This is a method for the generic function `intensity` It computes the empirical intensity of a point pattern on a linear network (object of class "lpp"), i.e. the average density of points per unit length.

If the point pattern is multitype, the intensities of the different types are computed separately.

Value

A numeric value (giving the intensity) or numeric vector (giving the intensity for each possible type).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`intensity`, `intensity.ppp`

Examples

```
intensity(chicago)
```

`intensity.ppm` *Intensity of Fitted Point Process Model*

Description

Computes the intensity of a fitted point process model.

Usage

```
## S3 method for class 'ppm'
intensity(X, ...)
```

Arguments

`X`	A fitted point process model (object of class "ppm").
`...`	Arguments passed to `predict.ppm` in some cases. See Details.

Details

This is a method for the generic function `intensity` for fitted point process models (class "ppm").

The intensity of a point process model is the expected number of random points per unit area.

If `X` is a Poisson point process model, the intensity of the process is computed exactly. The result is a numerical value if `X` is a stationary Poisson point process, and a pixel image if `X` is non-stationary. (In the latter case, the resolution of the pixel image is controlled by the arguments `...` which are passed to `predict.ppm`.)

If `X` is another Gibbs point process model, the intensity is computed approximately using the Poisson-saddlepoint approximation (Baddeley and Nair, 2012a, 2012b, 2016; Anderssen et al, 2014). The approximation is currently available for pairwise-interaction models (Baddeley and Nair, 2012a, 2012b) and for the area-interaction model and Geyer saturation model (Baddeley and Nair, 2016).

For a non-stationary Gibbs model, the pseudostationary solution (Baddeley and Nair, 2012b; Anderssen et al, 2014) is used. The result is a pixel image, whose resolution is controlled by the arguments ... which are passed to `predict.ppm`.

Value

A numeric value (if the model is stationary) or a pixel image.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Gopalan Nair.

References

Anderssen, R.S., Baddeley, A., DeHoog, F.R. and Nair, G.M. (2014) Solution of an integral equation arising in spatial point process theory. *Journal of Integral Equations and Applications* **26** (4) 437–453.

Baddeley, A. and Nair, G. (2012a) Fast approximation of the intensity of Gibbs point processes. *Electronic Journal of Statistics* **6** 1155–1169.

Baddeley, A. and Nair, G. (2012b) Approximating the moments of a spatial point process. *Stat* **1**, 1, 18–30. doi: 10.1002/sta4.5

Baddeley, A. and Nair, G. (2016) Poisson-saddlepoint approximation for spatial point processes with infinite order interaction. Submitted for publication.

See Also

`intensity`, `intensity.ppp`

Examples

```
fitP <- ppm(swedishpines ~ 1)
intensity(fitP)
fitS <- ppm(swedishpines ~ 1, Strauss(9))
intensity(fitS)
fitSx <- ppm(swedishpines ~ x, Strauss(9))
lamSx <- intensity(fitSx)
fitG <- ppm(swedishpines ~ 1, Geyer(9, 1))
lamG <- intensity(fitG)
fitA <- ppm(swedishpines ~ 1, AreaInter(7))
lamA <- intensity(fitA)
```

intensity.ppp *Empirical Intensity of Point Pattern*

Description

Computes the average number of points per unit area in a point pattern dataset.

Usage

```
## S3 method for class 'ppp'
intensity(X, ..., weights=NULL)

## S3 method for class 'splitppp'
intensity(X, ..., weights=NULL)
```

Arguments

X	A point pattern (object of class "ppp").
weights	Optional. Numeric vector of weights attached to the points of X. Alternatively, an expression which can be evaluated to give a vector of weights.
...	Ignored.

Details

This is a method for the generic function intensity. It computes the empirical intensity of a point pattern (object of class "ppp"), i.e. the average density of points per unit area.

If the point pattern is multitype, the intensities of the different types are computed separately.

Note that the intensity will be computed as the number of points per square unit, based on the unit of length for X, given by unitname(X). If the unit of length is a strange multiple of a standard unit, like 5.7 metres, then it can be converted to the standard unit using rescale. See the Examples.

If weights are given, then the intensity is computed as the total *weight* per square unit. The argument weights should be a numeric vector of weights for each point of X (weights may be negative or zero).

Alternatively weights can be an expression which will be evaluated for the dataset to yield a vector of weights. The expression may involve the Cartesian coordinates x, y of the points, and the marks of the points, if any. Variable names permitted in the expression include x and y, the name marks if X has a single column of marks, the names of any columns of marks if X has a data frame of marks, and the names of constants or functions that exist in the global environment. See the Examples.

Value

A numeric value (giving the intensity) or numeric vector (giving the intensity for each possible type).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

intensity, intensity.ppm

Examples

```
japanesepines
intensity(japanesepines)
unitname(japanesepines)
intensity(rescale(japanesepines))

intensity(amacrine)
intensity(split(amacrine))

# numeric vector of weights
volumes <- with(marks(finpines), (pi/4) * height * diameter^2)
intensity(finpines, weights=volumes)

# expression for weights
intensity(finpines, weights=expression((pi/4) * height * diameter^2))
```

---

intensity.ppx *Intensity of a Multidimensional Space-Time Point Pattern*

Description

Calculates the intensity of points in a multi-dimensional point pattern of class "ppx" or "pp3".

Usage

```
## S3 method for class 'ppx'
intensity(X, ...)
```

Arguments

X	Point pattern of class "ppx" or "pp3".
...	Ignored.

Details

This is a method for the generic function intensity. It computes the empirical intensity of a multi-dimensional point pattern (object of class "ppx" including "pp3"), i.e. the average density of points per unit volume.

If the point pattern is multitype, the intensities of the different types are computed separately.

Value

A single number or a numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
X <- osteo$pts[[1]]
intensity(X)
marks(X) <- factor(sample(letters[1:3], npoints(X), replace=TRUE))
intensity(X)
```

`intensity.psp` *Empirical Intensity of Line Segment Pattern*

Description

Computes the average total length of segments per unit area in a spatial pattern of line segments.

Usage

```
## S3 method for class 'psp'
intensity(X, ..., weights=NULL)
```

Arguments

`X`	A line segment pattern (object of class "psp").
`weights`	Optional. Numeric vector of weights attached to the segments of `X`. Alternatively, an `expression` which can be evaluated to give a vector of weights.
`...`	Ignored.

Details

This is a method for the generic function `intensity`. It computes the empirical intensity of a line segment pattern (object of class "psp"), i.e. the average total segment length per unit area.

If the segment pattern is multitype, the intensities of the different types are computed separately.

Note that the intensity will be computed as the length per area in units per square unit, based on the unit of length for `X`, given by `unitname(X)`. If the unit of length is a strange multiple of a standard unit, like `5.7 metres`, then it can be converted to the standard unit using `rescale`. See the Examples.

If `weights` are given, then the intensity is computed as the total *weight times length* per square unit. The argument `weights` should be a numeric vector of weights for each point of `X` (weights may be negative or zero).

Alternatively `weights` can be an `expression` which will be evaluated for the dataset to yield a vector of weights. The expression may involve the Cartesian coordinates x, y of the points, and the marks of the points, if any. Variable names permitted in the expression include `x0`, `x1`, `y0`, `y1` for the coordinates of the segment endpoint, the name `marks` if `X` has a single column of marks, the names of any columns of marks if `X` has a data frame of marks, and the names of constants or functions that exist in the global environment. See the Examples.

Value

A numeric value (giving the intensity) or numeric vector (giving the intensity for each possible type).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`intensity`

Examples

```
S <- as.psp(simplenet)
intensity(S)
intensity(S, weights=runif(nsegments(S)))
intensity(S, weights=expression((x0+x1)/2))
```

`intensity.quadratcount` *Intensity Estimates Using Quadrat Counts*

Description

Uses quadrat count data to estimate the intensity of a point pattern in each tile of a tessellation, assuming the intensity is constant in each tile.

Usage

```
## S3 method for class 'quadratcount'
intensity(X, ..., image=FALSE)
```

Arguments

`X`	An object of class `"quadratcount"`.
`image`	Logical value specifying whether to return a table of estimated intensities (the default) or a pixel image of the estimated intensity (`image=TRUE`).
`...`	Arguments passed to `as.mask` to determine the resolution of the pixel image, if `image=TRUE`.

Details

This is a method for the generic function `intensity`. It computes an estimate of the intensity of a point pattern from its quadrat counts.

The argument `X` should be an object of class `"quadratcount"`. It would have been obtained by applying the function `quadratcount` to a point pattern (object of class `"ppp"`). It contains the counts of the numbers of points of the point pattern falling in each tile of a tessellation.

Using this information, `intensity.quadratcount` divides the quadrat counts by the tile areas, yielding the average density of points per unit area in each tile of the tessellation.

If `image=FALSE` (the default), these intensity values are returned in a contingency table. Cells of the contingency table correspond to tiles of the tessellation.

If `image=TRUE`, the estimated intensity function is returned as a pixel image. For each pixel, the pixel value is the estimated intensity in the tile which contains that pixel.

Value

If `image=FALSE` (the default), a contingency table. If `image=TRUE`, a pixel image (object of class `"im"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`intensity`, `quadratcount`

Examples

```
qa <- quadratcount(swedishpines, 4,3)
qa
intensity(qa)
plot(intensity(qa, image=TRUE))
```

`interp.colourmap` *Interpolate smoothly between specified colours*

Description

Given a colourmap object which maps numbers to colours, this function interpolates smoothly between the colours, yielding a new colour map.

Usage

```
interp.colourmap(m, n = 512)
```

Arguments

`m`	A colour map (object of class `"colourmap"`).
`n`	Number of colour steps to be created in the new colour map.

Details

Given a colourmap object `m`, which maps numerical values to colours, this function interpolates the mapping, yielding a new colour map.

This makes it easy to build a colour map that has smooth gradation between different colours or shades. First specify a small vector of numbers `x` which should be mapped to specific colours `y`. Use `m <- colourmap(y, inputs=x)` to create a colourmap that represents this simple mapping. Then apply `interp.colourmap(m)` to obtain a smooth transition between these points.

Value

Another colour map (object of class `"colourmap"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

colourmap, tweak.colourmap, colourtools.

Examples

```
co <- colourmap(inputs=c(0, 0.5, 1), c("black", "red", "white"))
plot(interp.colourmap(co))
```

---

interp.im *Interpolate a Pixel Image*

Description

Interpolates the values of a pixel image at any desired location in the frame.

Usage

```
interp.im(Z, x, y=NULL)
```

Arguments

Z	Pixel image (object of class "im") with numeric or integer values.
x,y	Vectors of Cartesian coordinates. Alternatively x can be a point pattern and y can be missing.

Details

A value at each location (x[i],y[i]) will be interpolated using the pixel values of Z at the four surrounding pixel centres, by simple bilinear interpolation.

At the boundary (where (x[i],y[i]) is not surrounded by four pixel centres) the value at the nearest pixel is taken.

The arguments x,y can be anything acceptable to xy.coords.

Value

Vector of interpolated values, with NA for points that lie outside the domain of the image.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
opa <- par(mfrow=c(1,2))
# coarse image
V <- as.im(function(x,y) { x^2 + y }, owin(), dimyx=10)
plot(V, main="coarse image", col=terrain.colors(256))

# lookup value at location (0.5,0.5)
V[list(x=0.5,y=0.5)]
# interpolated value at location (0.5,0.5)
interp.im(V, 0.5, 0.5)
# true value is 0.75

# how to obtain an interpolated image at a desired resolution
U <- as.im(interp.im, W=owin(), Z=V, dimyx=256)
plot(U, main="interpolated image", col=terrain.colors(256))
par(opa)
```

---

intersect.owin *Intersection, Union or Set Subtraction of Windows*

Description

Yields the intersection, union or set subtraction of windows.

Usage

```
intersect.owin(..., fatal=TRUE, p)
union.owin(..., p)
setminus.owin(A, B, ..., p)
```

Arguments

`A,B`	Windows (objects of class "owin").
`...`	Windows, or arguments passed to `as.mask` to control the discretisation.
`fatal`	Logical. Determines what happens if the intersection is empty.
`p`	Optional list of parameters passed to `polyclip` to control the accuracy of polygon geometry.

Details

The function `intersect.owin` computes the intersection between the windows given in `...`, while `union.owin` computes their union. The function `setminus.owin` computes the intersection of `A` with the complement of `B`.

For `intersect.owin` and `union.owin`, the arguments `...` must be either

- window objects of class "owin",
- data that can be coerced to this class by `as.owin`),
- lists of windows, of class "solist",
- named arguments of `as.mask` to control the discretisation if required.

For `setminus.owin`, the arguments `...` must be named arguments of `as.mask`.

If the intersection is empty, then if `fatal=FALSE` the result is NULL, while if `fatal=TRUE` an error occurs.

Value

A window (object of class "owin") or possibly NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

is.subset.owin, overlap.owin, boundingbox, owin.object

Examples

```
# rectangles
   u <- unit.square()
   v <- owin(c(0.5,3.5), c(0.4,2.5))
# polygon
   data(letterR)
# mask
   m <- as.mask(letterR)

# two rectangles
   intersect.owin(u, v)
   union.owin(u,v)
   setminus.owin(u,v)

# polygon and rectangle
   intersect.owin(letterR, v)
   union.owin(letterR,v)
   setminus.owin(letterR,v)

# mask and rectangle
   intersect.owin(m, v)
   union.owin(m,v)
   setminus.owin(m,v)

# mask and polygon
   p <- rotate(v, 0.2)
   intersect.owin(m, p)
   union.owin(m,p)
   setminus.owin(m,p)

# two polygons
   A <- letterR
   B <- rotate(letterR, 0.2)
   plot(boundingbox(A,B), main="intersection")
   w <- intersect.owin(A, B)
   plot(w, add=TRUE, col="lightblue")
   plot(A, add=TRUE)
   plot(B, add=TRUE)

   plot(boundingbox(A,B), main="union")
   w <- union.owin(A,B)
```

```
    plot(w, add=TRUE, col="lightblue")
    plot(A, add=TRUE)
    plot(B, add=TRUE)

    plot(boundingbox(A,B), main="set minus")
    w <- setminus.owin(A,B)
    plot(w, add=TRUE, col="lightblue")
    plot(A, add=TRUE)
    plot(B, add=TRUE)

 # intersection and union of three windows
    C <- shift(B, c(0.2, 0.3))
    plot(union.owin(A,B,C))
    plot(intersect.owin(A,B,C))
```

---

intersect.tess *Intersection of Two Tessellations*

Description

Yields the intersection of two tessellations, or the intersection of a tessellation with a window.

Usage

```
intersect.tess(X, Y, ..., keepmarks=FALSE)
```

Arguments

X,Y	Two tessellations (objects of class "tess"), or windows (objects of class "tess"), or other data that can be converted to tessellations by as.tess.
...	Optional arguments passed to as.mask to control the discretisation, if required.
keepmarks	Logical value. If TRUE, the marks attached to the tiles of X and Y will be retained as marks of the intersection tiles.

Details

A tessellation is a collection of disjoint spatial regions (called *tiles*) that fit together to form a larger spatial region. See tess.

If X and Y are not tessellations, they are first converted into tessellations by as.tess.

The function intersect.tess then computes the intersection between the two tessellations. This is another tessellation, each of whose tiles is the intersection of a tile from X and a tile from Y.

One possible use of this function is to slice a window W into subwindows determined by a tessellation. See the Examples.

Value

A tessellation (object of class "tess").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

tess, as.tess, intersect.owin

Examples

```
  opa <- par(mfrow=c(1,3))
# polygon
  data(letterR)
  plot(letterR)
# tessellation of rectangles
  X <- tess(xgrid=seq(2, 4, length=10), ygrid=seq(0, 3.5, length=8))
  plot(X)
  plot(intersect.tess(X, letterR))

  A <- runifpoint(10)
  B <- runifpoint(10)
  plot(DA <- dirichlet(A))
  plot(DB <- dirichlet(B))
  plot(intersect.tess(DA, DB))
  par(opa)

  marks(DA) <- 1:10
  marks(DB) <- 1:10
  plot(Z <- intersect.tess(DA,DB, keepmarks=TRUE))
  mZ <- marks(Z)
  tZ <- tiles(Z)
  for(i in which(mZ[,1] == 3)) plot(tZ[[i]], add=TRUE, col="pink")
```

---

invoke.symbolmap *Plot Data Using Graphics Symbol Map*

Description

Apply a graphics symbol map to a vector of data values and plot the resulting symbols.

Usage

```
invoke.symbolmap(map, values, x=NULL, y = NULL, ..., add = FALSE,
                 do.plot = TRUE, started = add && do.plot)
```

Arguments

map	Graphics symbol map (object of class "symbolmap").
values	Vector of data that can be mapped by the symbol map.
x,y	Coordinate vectors for the spatial locations of the symbols to be plotted.
...	Additional graphics parameters.
add	Logical value indicating whether to add the symbols to an existing plot (add=TRUE) or to initialise a new plot (add=FALSE, the default).
do.plot	Logical value indicating whether to actually perform the plotting.
started	Logical value indicating whether the plot has already been initialised.

Details

A symbol map is an association between data values and graphical symbols.

This command applies the symbol map `map` to the data `values` and plots the resulting symbols at the locations given by `xy.coords(x,y)`.

Value

(Invisibly) the maximum diameter of the symbols, in user coordinate units.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

See Also

`plot.symbolmap` to plot the graphics map itself.

`symbolmap` to create a graphics map.

Examples

```
g <- symbolmap(range=c(-1,1),
                 shape=function(x) ifelse(x > 0, "circles", "squares"),
                 size=function(x) sqrt(ifelse(x > 0, x/pi, -x))/15,
                 bg=function(x) ifelse(x > 0, "green", "red"))
plot(square(1), main="")
a <- invoke.symbolmap(g, runif(10, -1, 1), runifpoint(10), add=TRUE)
a
```

`iplot` *Point and Click Interface for Displaying Spatial Data*

Description

Plot spatial data with interactive (point-and-click) control over the plot.

Usage

```
iplot(x, ...)

 ## S3 method for class 'ppp'
iplot(x, ..., xname)

 ## S3 method for class 'linnet'
iplot(x, ..., xname)

 ## S3 method for class 'lpp'
iplot(x, ..., xname)

 ## S3 method for class 'layered'
```

```
iplot(x, ..., xname, visible)

 ## Default S3 method:
iplot(x, ..., xname)
```

Arguments

x	The spatial object to be plotted. An object of class "ppp", "psp", "im", "owin", "linnet", "lpp" or "layered".
...	Ignored.
xname	Optional. Character string to use as the title of the dataset.
visible	Optional. Logical vector indicating which layers of x should initially be turned on (visible).

Details

The function iplot generates a plot of the spatial dataset x and allows interactive control over the appearance of the plot using a point-and-click interface.

The function iplot is generic, with methods for for point patterns (iplot.ppp), layered objects (iplot.layered) and a default method. The default method will handle objects of class "psp", "im" and "owin" at least.

A new popup window is launched. The spatial dataset x is displayed in the middle of the window using the appropriate plot method.

The left side of the window contains buttons and sliders allowing the user to change the plot parameters.

The right side of the window contains navigation controls for zooming (changing magnification), panning (shifting the field of view relative to the data), redrawing and exiting.

If the user clicks in the area where the point pattern is displayed, the field of view will be re-centred at the point that was clicked.

Value

NULL.

Package Dependence

This function requires the package **rpanel** to be loaded.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

istat

Examples

```
if(interactive() && require(rpanel)) {
   iplot(cells)
   iplot(amacrine)
   iplot(lansing)
   L <- layered(D=distmap(cells), P=cells,
         plotargs=list(list(ribbon=FALSE), list(pch=16)))
   iplot(L)
}
```

---

ippm *Fit Point Process Model Involving Irregular Trend Parameters*

Description

Experimental extension to ppm which finds optimal values of the irregular trend parameters in a point process model.

Usage

```
ippm(Q, ...,
          iScore=NULL,
          start=list(),
          covfunargs=start,
          nlm.args=list(stepmax=1/2),
          silent=FALSE,
          warn.unused=TRUE)
```

Arguments

`Q,...`	Arguments passed to ppm to fit the point process model.
`iScore`	Optional. A named list of R functions that compute the partial derivatives of the logarithm of the trend, with respect to each irregular parameter. See Details.
`start`	Named list containing initial values of the irregular parameters over which to optimise.
`covfunargs`	Argument passed to ppm. A named list containing values for *all* irregular parameters required by the covariates in the model. Must include all the parameters named in `start`.
`nlm.args`	Optional list of arguments passed to nlm to control the optimization algorithm.
`silent`	Logical. Whether to print warnings if the optimization algorithm fails to converge.
`warn.unused`	Logical. Whether to print a warning if some of the parameters in `start` are not used in the model.

Details

This function is an experimental extension to the point process model fitting command `ppm`. The extension allows the trend of the model to include irregular parameters, which will be maximised by a Newton-type iterative method, using `nlm`.

For the sake of explanation, consider a Poisson point process with intensity function $\lambda(u)$ at location u. Assume that

$$\lambda(u) = \exp(\alpha + \beta Z(u))\, f(u, \gamma)$$

where α, β, γ are parameters to be estimated, $Z(u)$ is a spatial covariate function, and f is some known function. Then the parameters α, β are called *regular* because they appear in a loglinear form; the parameter γ is called *irregular*.

To fit this model using `ippm`, we specify the intensity using the `trend` formula in the same way as usual for `ppm`. The trend formula is a representation of the log intensity. In the above example the log intensity is

$$\log \lambda(u) = \alpha + \beta Z(u) + \log f(u, \gamma)$$

So the model above would be encoded with the trend formula `~Z + offset(log(f))`. Note that the irregular part of the model is an *offset* term, which means that it is included in the log trend as it is, without being multiplied by another regular parameter.

The optimisation runs faster if we specify the derivative of $\log f(u, \gamma)$ with respect to γ. We call this the *irregular score*. To specify this, the user must write an R function that computes the irregular score for any value of γ at any location `(x,y)`.

Thus, to code such a problem,

1. The argument `trend` should define the log intensity, with the irregular part as an offset;
2. The argument `start` should be a list containing initial values of each of the irregular parameters;
3. The argument `iScore`, if provided, must be a list (with one entry for each entry of `start`) of functions with arguments `x,y,...`, that evaluate the partial derivatives of $\log f(u, \gamma)$ with respect to each irregular parameter.

The coded example below illustrates the model with two irregular parameters γ, δ and irregular term

$$f((x, y), (\gamma, \delta)) = 1 + \exp(\gamma - \delta x^3)$$

Arguments `...` passed to `ppm` may also include `interaction`. In this case the model is not a Poisson point process but a more general Gibbs point process; the trend formula `trend` determines the first-order trend of the model (the first order component of the conditional intensity), not the intensity.

Value

A fitted point process model (object of class `"ppm"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`ppm`, `profilepl`

Examples

```
nd <- 32


gamma0 <- 3
delta0 <- 5
POW <- 3
# Terms in intensity
Z <- function(x,y) { -2*y }
f <- function(x,y,gamma,delta) { 1 + exp(gamma - delta * x^POW) }
# True intensity
lamb <- function(x,y,gamma,delta) { 200 * exp(Z(x,y)) * f(x,y,gamma,delta) }
# Simulate realisation
lmax <- max(lamb(0,0,gamma0,delta0), lamb(1,1,gamma0,delta0))
set.seed(42)
X <- rpoispp(lamb, lmax=lmax, win=owin(), gamma=gamma0, delta=delta0)
# Partial derivatives of log f
DlogfDgamma <- function(x,y, gamma, delta) {
  topbit <- exp(gamma - delta * x^POW)
  topbit/(1 + topbit)
}
DlogfDdelta <- function(x,y, gamma, delta) {
  topbit <- exp(gamma - delta * x^POW)
  - (x^POW) * topbit/(1 + topbit)
}
# irregular score
Dlogf <- list(gamma=DlogfDgamma, delta=DlogfDdelta)
# fit model
ippm(X ~Z + offset(log(f)),
     covariates=list(Z=Z, f=f),
     iScore=Dlogf,
     start=list(gamma=1, delta=1),
     nlm.args=list(stepmax=1),
     nd=nd)
```

`is.connected` *Determine Whether an Object is Connected*

Description

Determine whether an object is topologically connected.

Usage

```
is.connected(X, ...)

## Default S3 method:
is.connected(X, ...)

## S3 method for class 'linnet'
is.connected(X, ...)
```

Arguments

X	A spatial object such as a pixel image (object of class "im"), a window (object of class "owin") or a linear network (object of class "linnet").
...	Arguments passed to connected to determine the connected components.

Details

The command `is.connected(X)` returns `TRUE` if the object `X` consists of a single, topologically-connected piece, and returns `FALSE` if `X` consists of several pieces which are not joined together.

The function `is.connected` is generic. The default method `is.connected.default` works for many classes of objects, including windows (class "owin") and images (class "im"). There is a method for linear networks, `is.connected.linnet`, described here, and a method for point patterns described in `is.connected.ppp`.

Value

A logical value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`connected`, `is.connected.ppp`.

Examples

```
d <- distmap(cells, dimyx=256)
X <- levelset(d, 0.07)
plot(X)
is.connected(X)
```

---

is.connected.ppp — *Determine Whether a Point Pattern is Connected*

Description

Determine whether a point pattern is topologically connected when all pairs of points closer than a threshold distance are joined.

Usage

```
## S3 method for class 'ppp'
is.connected(X, R, ...)
```

Arguments

X	A point pattern (object of class "ppp").
R	Threshold distance. Pairs of points closer than R units apart will be joined together.
...	Ignored.

Details

The function `is.connected` is generic. This is the method for point patterns (objects of class `"ppp"`).

The point pattern `X` is first converted into an abstract graph by joining every pair of points that lie closer than `R` units apart. Then the algorithm determines whether this graph is connected.

That is, the result of `is.connected(X)` is `TRUE` if any point in `X` can be reached from any other point, by a series of steps between points of `X`, each step being shorter than `R` units in length.

Value

A logical value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`is.connected`, `connected.ppp`.

Examples

```
is.connected(redwoodfull, 0.1)
is.connected(redwoodfull, 0.2)
```

`is.convex` *Test Whether a Window is Convex*

Description

Determines whether a window is convex.

Usage

```
is.convex(x)
```

Arguments

`x`	Window (object of class `"owin"`).

Details

If `x` is a rectangle, the result is TRUE.

If `x` is polygonal, the result is TRUE if `x` consists of a single polygon and this polygon is equal to the minimal convex hull of its vertices computed by `chull`.

If `x` is a mask, the algorithm first extracts all boundary pixels of `x` using `vertices`. Then it computes the (polygonal) convex hull K of the boundary pixels. The result is TRUE if every boundary pixel lies within one pixel diameter of an edge of K.

Value

Logical value, equal to TRUE if x is convex.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin, convexhull.xy, vertices

---

is.dppm *Recognise Fitted Determinantal Point Process Models*

Description

Check that an object inherits the class dppm

Usage

```
is.dppm(x)
```

Arguments

x	Any object.

Value

A single logical value.

Author(s)

Ege Rubak <rubak@math.aau.dk> <rubak@math.aau.dk>, Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <Adrian.Baddeley@uwa.edu.au> and Rolf Turner <r.turner@auckland.ac.nz> <r.turner@auckland.ac.nz>

---

is.empty *Test Whether An Object Is Empty*

Description

Checks whether the argument is an empty window, an empty point pattern, etc.

Usage

```
is.empty(x)
## S3 method for class 'owin'
is.empty(x)
## S3 method for class 'ppp'
is.empty(x)
## S3 method for class 'psp'
is.empty(x)
## Default S3 method:
is.empty(x)
```

Arguments

x	A window (object of class "owin"), a point pattern (object of class "ppp"), or a line segment pattern (object of class "psp").

Details

This function tests whether the object x represents an empty spatial object, such as an empty window, a point pattern with zero points, or a line segment pattern with zero line segments.

An empty window can be obtained as the output of intersect.owin, erosion, opening, complement.owin and some other operations.

An empty point pattern or line segment pattern can be obtained as the result of simulation.

Value

Logical value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

---

is.hybrid *Test Whether Object is a Hybrid*

Description

Tests where a point process model or point process interaction is a hybrid of several interactions.

Usage

```
is.hybrid(x)

## S3 method for class 'ppm'
is.hybrid(x)

## S3 method for class 'interact'
is.hybrid(x)
```

Arguments

x — A point process model (object of class "ppm") or a point process interaction structure (object of class "interact").

Details

A *hybrid* (Baddeley, Turner, Mateu and Bevan, 2012) is a point process model created by combining two or more point process models, or an interpoint interaction created by combining two or more interpoint interactions.

The function `is.hybrid` is generic, with methods for point process models (objects of class "ppm") and point process interactions (objects of class "interact"). These functions return `TRUE` if the object `x` is a hybrid, and `FALSE` if it is not a hybrid.

Hybrids of two or more interpoint interactions are created by the function `Hybrid`. Such a hybrid interaction can then be fitted to point pattern data using `ppm`.

Value

`TRUE` if the object is a hybrid, and `FALSE` otherwise.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Turner, R., Mateu, J. and Bevan, A. (2013) Hybrids of Gibbs point process models and their implementation. *Journal of Statistical Software* **55**:11, 1–43. http://www.jstatsoft.org/v55/i11/

See Also

`Hybrid`

Examples

```
S <- Strauss(0.1)
is.hybrid(S)
H <- Hybrid(Strauss(0.1), Geyer(0.2, 3))
is.hybrid(H)

data(redwood)
fit <- ppm(redwood, ~1, H)
is.hybrid(fit)
```

`is.im` *Test Whether An Object Is A Pixel Image*

Description

Tests whether its argument is a pixel image (object of class "im").

Usage

```
is.im(x)
```

Arguments

x	Any object.

Details

This function tests whether the argument x is a pixel image object of class "im". For details of this class, see im.object.

The object is determined to be an image if it inherits from class "im".

Value

TRUE if x is a pixel image, otherwise FALSE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

`is.lpp` *Test Whether An Object Is A Point Pattern on a Linear Network*

Description

Checks whether its argument is a point pattern on a linear network (object of class "lpp").

Usage

```
is.lpp(x)
```

Arguments

x	Any object.

Details

This function tests whether the object x is a point pattern object of class "lpp".

Value

TRUE if x is a point pattern of class "lpp", otherwise FALSE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

---

is.marked *Test Whether Marks Are Present*

Description

Generic function to test whether a given object (usually a point pattern or something related to a point pattern) has "marks" attached to the points.

Usage

```
is.marked(X, ...)
```

Arguments

X	Object to be inspected
...	Other arguments.

Details

"Marks" are observations attached to each point of a point pattern. For example the longleaf dataset contains the locations of trees, each tree being marked by its diameter; the amacrine dataset gives the locations of cells of two types (on/off) and the type of cell may be regarded as a mark attached to the location of the cell.

Other objects related to point patterns, such as point process models, may involve marked points.

This function tests whether the object X contains or involves marked points. It is generic; methods are provided for point patterns (objects of class "ppp") and point process models (objects of class "ppm").

Value

Logical value, equal to TRUE if X is marked.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

is.marked.ppp, is.marked.ppm

is.marked.ppm *Test Whether A Point Process Model is Marked*

Description

Tests whether a fitted point process model involves “marks” attached to the points.

Usage

```
  ## S3 method for class 'ppm'
is.marked(X, ...)

  ## S3 method for class 'lppm'
is.marked(X, ...)
```

Arguments

X	Fitted point process model (object of class "ppm") usually obtained from ppm. Alternatively, a model of class "lppm".
...	Ignored.

Details

“Marks” are observations attached to each point of a point pattern. For example the longleaf dataset contains the locations of trees, each tree being marked by its diameter; the amacrine dataset gives the locations of cells of two types (on/off) and the type of cell may be regarded as a mark attached to the location of the cell.

The argument X is a fitted point process model (an object of class "ppm") typically obtained by fitting a model to point pattern data using ppm.

This function returns TRUE if the *original data* (to which the model X was fitted) were a marked point pattern.

Note that this is not the same as testing whether the model involves terms that depend on the marks (i.e. whether the fitted model ignores the marks in the data). Currently we have not implemented a test for this.

If this function returns TRUE, the implications are (for example) that any simulation of this model will require simulation of random marks as well as random point locations.

Value

Logical value, equal to TRUE if X is a model that was fitted to a marked point pattern dataset.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

is.marked, is.marked.ppp

Examples

```
 X <- lansing
 # Multitype point pattern --- trees marked by species



fit1 <- ppm(X, ~ marks, Poisson())
is.marked(fit1)
# TRUE

fit2 <- ppm(X, ~ 1, Poisson())
is.marked(fit2)
# TRUE

# Unmarked point pattern
fit3 <- ppm(cells, ~ 1, Poisson())
is.marked(fit3)
# FALSE
```

`is.marked.ppp` *Test Whether A Point Pattern is Marked*

Description

Tests whether a point pattern has "marks" attached to the points.

Usage

```
  ## S3 method for class 'ppp'
is.marked(X, na.action="warn", ...)
```

Arguments

`X`	Point pattern (object of class "ppp")
`na.action`	String indicating what to do if `NA` values are encountered amongst the marks. Options are `"warn"`, `"fatal"` and `"ignore"`.
`...`	Ignored.

Details

"Marks" are observations attached to each point of a point pattern. For example the `longleaf` dataset contains the locations of trees, each tree being marked by its diameter; the `amacrine` dataset gives the locations of cells of two types (on/off) and the type of cell may be regarded as a mark attached to the location of the cell.

This function tests whether the point pattern `X` contains or involves marked points. It is a method for the generic function `is.marked`.

The argument `na.action` determines what action will be taken if the point pattern has a vector of marks but some or all of the marks are `NA`. Options are `"fatal"` to cause a fatal error; `"warn"` to issue a warning and then return `TRUE`; and `"ignore"` to take no action except returning `TRUE`.

Value

Logical value, equal to TRUE if X is a marked point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

is.marked, is.marked.ppm

Examples

```
data(cells)
is.marked(cells)  #FALSE
data(longleaf)
is.marked(longleaf) #TRUE
```

---

is.multitype *Test whether Object is Multitype*

Description

Generic function to test whether a given object (usually a point pattern or something related to a point pattern) has "marks" attached to the points which classify the points into several types.

Usage

```
is.multitype(X, ...)
```

Arguments

X	Object to be inspected
...	Other arguments.

Details

"Marks" are observations attached to each point of a point pattern. For example the longleaf dataset contains the locations of trees, each tree being marked by its diameter; the amacrine dataset gives the locations of cells of two types (on/off) and the type of cell may be regarded as a mark attached to the location of the cell. Other objects related to point patterns, such as point process models, may involve marked points.

This function tests whether the object X contains or involves marked points, **and** that the marks are a factor.

For example, the amacrine dataset is multitype (there are two types of cells, on and off), but the longleaf dataset is *not* multitype (the marks are real numbers).

This function is generic; methods are provided for point patterns (objects of class "ppp") and point process models (objects of class "ppm").

Value

Logical value, equal to TRUE if X is multitype.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

is.multitype.ppp, is.multitype.ppm

---

is.multitype.ppm *Test Whether A Point Process Model is Multitype*

Description

Tests whether a fitted point process model involves “marks” attached to the points that classify the points into several types.

Usage

```
  ## S3 method for class 'ppm'
is.multitype(X, ...)

  ## S3 method for class 'lppm'
is.multitype(X, ...)
```

Arguments

X	Fitted point process model (object of class "ppm") usually obtained from ppm. Alternatively a model of class "lppm".
...	Ignored.

Details

“Marks” are observations attached to each point of a point pattern. For example the longleaf dataset contains the locations of trees, each tree being marked by its diameter; the amacrine dataset gives the locations of cells of two types (on/off) and the type of cell may be regarded as a mark attached to the location of the cell.

The argument X is a fitted point process model (an object of class "ppm") typically obtained by fitting a model to point pattern data using ppm.

This function returns TRUE if the *original data* (to which the model X was fitted) were a multitype point pattern.

Note that this is not the same as testing whether the model involves terms that depend on the marks (i.e. whether the fitted model ignores the marks in the data). Currently we have not implemented a test for this.

If this function returns TRUE, the implications are (for example) that any simulation of this model will require simulation of random marks as well as random point locations.

Value

Logical value, equal to TRUE if X is a model that was fitted to a multitype point pattern dataset.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

is.multitype, is.multitype.ppp

Examples

```
X <- lansing
# Multitype point pattern --- trees marked by species



fit1 <- ppm(X, ~ marks, Poisson())
is.multitype(fit1)
# TRUE

fit2 <- ppm(X, ~ 1, Poisson())
is.multitype(fit2)
# TRUE

# Unmarked point pattern
fit3 <- ppm(cells, ~ 1, Poisson())
is.multitype(fit3)
# FALSE
```

---

is.multitype.ppp *Test Whether A Point Pattern is Multitype*

Description

Tests whether a point pattern has "marks" attached to the points which classify the points into several types.

Usage

```
## S3 method for class 'ppp'
is.multitype(X, na.action="warn", ...)

## S3 method for class 'lpp'
is.multitype(X, na.action="warn", ...)
```

Arguments

`X`	Point pattern (object of class "ppp" or "lpp")
`na.action`	String indicating what to do if `NA` values are encountered amongst the marks. Options are "warn", "fatal" and "ignore".
`...`	Ignored.

Details

"Marks" are observations attached to each point of a point pattern. For example the `longleaf` dataset contains the locations of trees, each tree being marked by its diameter; the `amacrine` dataset gives the locations of cells of two types (on/off) and the type of cell may be regarded as a mark attached to the location of the cell.

This function tests whether the point pattern `X` contains or involves marked points, **and** that the marks are a factor. It is a method for the generic function `is.multitype`.

For example, the `amacrine` dataset is multitype (there are two types of cells, on and off), but the `longleaf` dataset is *not* multitype (the marks are real numbers).

The argument `na.action` determines what action will be taken if the point pattern has a vector of marks but some or all of the marks are `NA`. Options are "fatal" to cause a fatal error; "warn" to issue a warning and then return `TRUE`; and "ignore" to take no action except returning `TRUE`.

Value

Logical value, equal to `TRUE` if `X` is a multitype point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`is.multitype`, `is.multitype.ppm`

Examples

```
is.multitype(cells)  #FALSE - no marks
is.multitype(longleaf) #FALSE - real valued marks
is.multitype(amacrine) #TRUE
```

---

is.owin — *Test Whether An Object Is A Window*

Description

Checks whether its argument is a window (object of class "owin").

Usage

```
is.owin(x)
```

Arguments

x Any object.

Details

This function tests whether the object x is a window object of class "owin". See owin.object for details of this class.

The result is determined to be TRUE if x inherits from "owin", i.e. if x has "owin" amongst its classes.

Value

TRUE if x is a point pattern, otherwise FALSE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

---

is.ppm *Test Whether An Object Is A Fitted Point Process Model*

Description

Checks whether its argument is a fitted point process model (object of class "ppm", "kppm", "lppm" or "slrm").

Usage

```
is.ppm(x)
is.kppm(x)
is.lppm(x)
is.slrm(x)
```

Arguments

x Any object.

Details

These functions test whether the object x is a fitted point process model object of the specified class.

The result of is.ppm(x) is TRUE if x has "ppm" amongst its classes, and otherwise FALSE. Similarly for the other functions.

Value

A single logical value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

is.ppp	*Test Whether An Object Is A Point Pattern*

Description

Checks whether its argument is a point pattern (object of class "ppp").

Usage

```
is.ppp(x)
```

Arguments

x	Any object.

Details

This function tests whether the object x is a point pattern object of class "ppp". See ppm.object for details of this class.

The result is determined to be TRUE if x inherits from "ppp", i.e. if x has "ppp" amongst its classes.

Value

TRUE if x is a point pattern, otherwise FALSE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

is.rectangle	*Determine Type of Window*

Description

Determine whether a window is a rectangle, a polygonal region, or a binary mask.

Usage

```
is.rectangle(w)
is.polygonal(w)
is.mask(w)
```

Arguments

w	Window to be inspected. An object of class "owin".

Details

These simple functions determine whether a window w (object of class "owin") is a rectangle (is.rectangle(w) = TRUE), a domain with polygonal boundary (is.polygonal(w) = TRUE), or a binary pixel mask (is.mask(w) = TRUE).

Value

Logical value, equal to TRUE if w is a window of the specified type.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin

---

is.stationary *Recognise Stationary and Poisson Point Process Models*

Description

Given a point process model that has been fitted to data, determine whether the model is a stationary point process, and whether it is a Poisson point process.

Usage

```
is.stationary(x)
## S3 method for class 'ppm'
is.stationary(x)
## S3 method for class 'kppm'
is.stationary(x)
## S3 method for class 'lppm'
is.stationary(x)
## S3 method for class 'slrm'
is.stationary(x)
## S3 method for class 'rmhmodel'
is.stationary(x)
## S3 method for class 'dppm'
is.stationary(x)
## S3 method for class 'detpointprocfamily'
is.stationary(x)

is.poisson(x)
## S3 method for class 'ppm'
is.poisson(x)
## S3 method for class 'kppm'
is.poisson(x)
## S3 method for class 'lppm'
is.poisson(x)
```

```
## S3 method for class 'slrm'
is.poisson(x)
## S3 method for class 'rmhmodel'
is.poisson(x)
## S3 method for class 'interact'
is.poisson(x)
```

Arguments

x	A fitted spatial point process model (object of class "ppm", "kppm", "lppm", "dppm" or "slrm") or similar object.

Details

The argument x represents a fitted spatial point process model or a similar object.

is.stationary(x) returns TRUE if x represents a stationary point process, and FALSE if not.

is.poisson(x) returns TRUE if x represents a Poisson point process, and FALSE if not.

The functions is.stationary and is.poisson are generic, with methods for the classes "ppm" (Gibbs point process models), "kppm" (cluster or Cox point process models), "slrm" (spatial logistic regression models) and "rmhmodel" (model specifications for the Metropolis-Hastings algorithm). Additionally is.stationary has a method for classes "detpointprocfamily" and "dppm" (both determinantal point processes) and is.poisson has a method for class "interact" (interaction structures for Gibbs models).

is.poisson.kppm will return FALSE, unless the model x is degenerate: either x has zero intensity so that its realisations are empty with probability 1, or it is a log-Gaussian Cox process where the log intensity has zero variance.

is.poisson.slrm will always return TRUE, by convention.

Value

A logical value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

is.marked to determine whether a model is a marked point process.

summary.ppm for detailed information.

Model-fitting functions ppm, dppm, kppm, lppm, slrm.

Examples

```
data(cells)
data(redwood)

fit <- ppm(cells ~ x)
is.stationary(fit)
```

```
is.poisson(fit)

fut <- kppm(redwood ~ 1, "MatClust")
is.stationary(fut)
is.poisson(fut)

fot <- slrm(cells ~ x)
is.stationary(fot)
is.poisson(fot)
```

---

is.subset.owin *Determine Whether One Window is Contained In Another*

Description

Tests whether window A is a subset of window B.

Usage

```
is.subset.owin(A, B)
```

Arguments

A	A window object (see Details).
B	A window object (see Details).

Details

This function tests whether the window A is a subset of the window B.

The arguments A and B must be window objects (either objects of class "owin", or data that can be coerced to this class by as.owin).

Various algorithms are used, depending on the geometrical type of the two windows.

Note that if B is not rectangular, the algorithm proceeds by discretising A, converting it to a pixel mask using as.mask. In this case the resulting answer is only “approximately correct”. The accuracy of the approximation can be controlled: see as.mask.

Value

Logical scalar; TRUE if A is a sub-window of B, otherwise FALSE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
w1 <- as.owin(c(0,1,0,1))
w2 <- as.owin(c(-1,2,-1,2))
is.subset.owin(w1,w2)  # Returns TRUE.
is.subset.owin(w2,w1)  # Returns FALSE.
```

`istat` *Point and Click Interface for Exploratory Analysis of Point Pattern*

Description

Compute various summary functions for a point pattern using a point-and-click interface.

Usage

```
istat(x, xname)
```

Arguments

`x`	The spatial point pattern to be analysed. An object of class `"ppp"`.
`xname`	Optional. Character string to use as the title of the dataset.

Details

This command launches an interactive (point-and-click) interface which offers a choice of spatial summary functions that can be applied to the point pattern `x`.

The selected summary function is computed for the point pattern `x` and plotted in a popup window.

The selection of functions includes `Kest`, `Lest`, `pcf`, `Fest` ,`Gest` and `Jest`. For the function `pcf` it is possible to control the bandwidth parameter `bw`.

There is also an option to show simulation envelopes of the summary function.

Value

`NULL`.

Note

Before adjusting the bandwidth parameter `bw`, it is advisable to select *No simulation envelopes* to save a lot of computation time.

Package Dependence

This function requires the package **rpanel** to be loaded.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`iplot`

Examples

```
if(interactive() && require(rpanel)) {
   istat(swedishpines)
}
```

Jcross	*Multitype J Function (i-to-j)*

Description

For a multitype point pattern, estimate the multitype J function summarising the interpoint dependence between points of type i and of type j.

Usage

```
Jcross(X, i, j, eps=NULL, r=NULL, breaks=NULL, ..., correction=NULL)
```

Arguments

`X`	The observed point pattern, from which an estimate of the multitype J function $J_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
`i`	The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.
`j`	The type (mark value) of the points in `X` to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of `marks(X)`.
`eps`	A positive number. The resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
`r`	Optional. Numeric vector. The values of the argument r at which the function $J_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`breaks`	This argument is for internal use only.
`...`	Ignored.
`correction`	Optional. Character string specifying the edge correction(s) to be used. Options are `"none"`, `"rs"`, `"km"`, `"Hanisch"` and `"best"`. Alternatively `correction="all"` selects all options.

Details

This function `Jcross` and its companions `Jdot` and `Jmulti` are generalisations of the function `Jest` to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible "colours" or "types". In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument `X` must be a point pattern (object of class `"ppp"`) or any data that are acceptable to `as.ppp`. It must be a marked point pattern, and the mark vector `X$marks` must be a factor. The argument `i` will be interpreted as a level of the factor `X$marks`. (Warning: this means that an integer value `i=3` will be interpreted as the number 3, **not** the 3rd smallest level).

The "type i to type j" multitype J function of a stationary multitype point process X was introduced by Van lieshout and Baddeley (1999). It is defined by

$$J_{ij}(r) = \frac{1 - G_{ij}(r)}{1 - F_j(r)}$$

where $G_{ij}(r)$ is the distribution function of the distance from a type i point to the nearest point of type j, and $F_j(r)$ is the distribution function of the distance from a fixed point in space to the nearest point of type j in the pattern.

An estimate of $J_{ij}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the subprocess of type i points is independent of the subprocess of points of type j, then $J_{ij}(r) \equiv 1$. Hence deviations of the empirical estimate of J_{ij} from the value 1 may suggest dependence between types.

This algorithm estimates $J_{ij}(r)$ from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in `Jest`, using the Kaplan-Meier and border corrections. The main work is done by `Gmulti` and `Fest`.

The argument `r` is the vector of values for the distance r at which $J_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must exceed the radius of the largest disc contained in the window.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing six numeric columns

`J`	the recommended estimator of $J_{ij}(r)$, currently the Kaplan-Meier estimator.
`r`	the values of the argument r at which the function $J_{ij}(r)$ has been estimated
`km`	the Kaplan-Meier estimator of $J_{ij}(r)$
`rs`	the "reduced sample" or "border correction" estimator of $J_{ij}(r)$
`han`	the Hanisch-style estimator of $J_{ij}(r)$
`un`	the "uncorrected" estimator of $J_{ij}(r)$ formed by taking the ratio of uncorrected empirical estimators of $1 - G_{ij}(r)$ and $1 - F_j(r)$, see `Gdot` and `Fest`.
`theo`	the theoretical value of $J_{ij}(r)$ for a marked Poisson process, namely 1.

The result also has two attributes `"G"` and `"F"` which are respectively the outputs of `Gcross` and `Fest` for the point pattern.

Warnings

The arguments `i` and `j` are always interpreted as levels of the factor `X$marks`. They are converted to character strings if they are not already character strings. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

Jdot, Jest, Jmulti

Examples

```
 # Lansing woods data: 6 types of trees
woods <- lansing

Jhm <- Jcross(woods, "hickory", "maple")
# diagnostic plot for independence between hickories and maples
plot(Jhm)

# synthetic example with two types "a" and "b"
pp <- runifpoint(30) %mark% factor(sample(c("a","b"), 30, replace=TRUE))
J <- Jcross(pp)
```

`Jdot` *Multitype J Function (i-to-any)*

Description

For a multitype point pattern, estimate the multitype J function summarising the interpoint dependence between the type i points and the points of any type.

Usage

```
Jdot(X, i, eps=NULL, r=NULL, breaks=NULL, ..., correction=NULL)
```

Arguments

`X`	The observed point pattern, from which an estimate of the multitype J function $J_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
`i`	The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.
`eps`	A positive number. The resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
`r`	numeric vector. The values of the argument r at which the function $J_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`breaks`	This argument is for internal use only.
`...`	Ignored.

correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "Hanisch" and "best". Alternatively correction="all" selects all options.

Details

This function Jdot and its companions Jcross and Jmulti are generalisations of the function Jest to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible "colours" or "types". In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor. The argument i will be interpreted as a level of the factor X$marks. (Warning: this means that an integer value i=3 will be interpreted as the number 3, **not** the 3rd smallest level.)

The "type i to any type" multitype J function of a stationary multitype point process X was introduced by Van lieshout and Baddeley (1999). It is defined by

$$J_{i\bullet}(r) = \frac{1 - G_{i\bullet}(r)}{1 - F_{\bullet}(r)}$$

where $G_{i\bullet}(r)$ is the distribution function of the distance from a type i point to the nearest other point of the pattern, and $F_{\bullet}(r)$ is the distribution function of the distance from a fixed point in space to the nearest point of the pattern.

An estimate of $J_{i\bullet}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the pattern is a marked Poisson point process, then $J_{i\bullet}(r) \equiv 1$. If the subprocess of type i points is independent of the subprocess of points of all types not equal to i, then $J_{i\bullet}(r)$ equals $J_{ii}(r)$, the ordinary J function (see Jest and Van Lieshout and Baddeley (1996)) of the points of type i. Hence deviations from zero of the empirical estimate of $J_{i\bullet} - J_{ii}$ may suggest dependence between types.

This algorithm estimates $J_{i\bullet}(r)$ from the point pattern X. It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in Jest, using the Kaplan-Meier and border corrections. The main work is done by Gmulti and Fest.

The argument r is the vector of values for the distance r at which $J_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must exceed the radius of the largest disc contained in the window.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing six numeric columns

J	the recommended estimator of $J_{i\bullet}(r)$, currently the Kaplan-Meier estimator.
r	the values of the argument r at which the function $J_{i\bullet}(r)$ has been estimated
km	the Kaplan-Meier estimator of $J_{i\bullet}(r)$
rs	the "reduced sample" or "border correction" estimator of $J_{i\bullet}(r)$
han	the Hanisch-style estimator of $J_{i\bullet}(r)$

un the "uncorrected" estimator of $J_{i\bullet}(r)$ formed by taking the ratio of uncorrected empirical estimators of $1 - G_{i\bullet}(r)$ and $1 - F_{\bullet}(r)$, see `Gdot` and `Fest`.

theo the theoretical value of $J_{i\bullet}(r)$ for a marked Poisson process, namely 1.

The result also has two attributes `"G"` and `"F"` which are respectively the outputs of `Gdot` and `Fest` for the point pattern.

Warnings

The argument `i` is interpreted as a level of the factor `X$marks`. It is converted to a character string if it is not already a character string. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

`Jcross`, `Jest`, `Jmulti`

Examples

```
    # Lansing woods data: 6 types of trees
  woods <- lansing


   Jh. <- Jdot(woods, "hickory")
   plot(Jh.)
   # diagnostic plot for independence between hickories and other trees
   Jhh <- Jest(split(woods)$hickory)
   plot(Jhh, add=TRUE, legendpos="bottom")

   ## Not run:
   # synthetic example with two marks "a" and "b"
   pp <- runifpoint(30) %mark% factor(sample(c("a","b"), 30, replace=TRUE))
   J <- Jdot(pp, "a")

## End(Not run)
```

Jest *Estimate the J-function*

Description

Estimates the summary function $J(r)$ for a point pattern in a window of arbitrary shape.

Usage

```
Jest(X, ..., eps=NULL, r=NULL, breaks=NULL, correction=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of $J(r)$ will be computed. An object of class "ppp", or data in any format acceptable to `as.ppp()`.
...	Ignored.
eps	the resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
r	vector of values for the argument r at which $J(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on `r`.
breaks	This argument is for internal use only.
correction	Optional. Character string specifying the choice of edge correction(s) in `Fest` and `Gest`. See Details.

Details

The J function (Van Lieshout and Baddeley, 1996) of a stationary point process is defined as

$$J(r) = \frac{1 - G(r)}{1 - F(r)}$$

where $G(r)$ is the nearest neighbour distance distribution function of the point process (see `Gest`) and $F(r)$ is its empty space function (see `Fest`).

For a completely random (uniform Poisson) point process, the J-function is identically equal to 1. Deviations $J(r) < 1$ or $J(r) > 1$ typically indicate spatial clustering or spatial regularity, respectively. The J-function is one of the few characteristics that can be computed explicitly for a wide range of point processes. See Van Lieshout and Baddeley (1996), Baddeley et al (2000), Thonnes and Van Lieshout (1999) for further information.

An estimate of J derived from a spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern. The estimate of $J(r)$ is compared against the constant function 1. Deviations $J(r) < 1$ or $J(r) > 1$ may suggest spatial clustering or spatial regularity, respectively.

This algorithm estimates the J-function from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape.

The argument `X` is interpreted as a point pattern object (of class "ppp", see `ppp.object`) and can be supplied in any of the formats recognised by `as.ppp()`.

The functions `Fest` and `Gest` are called to compute estimates of $F(r)$ and $G(r)$ respectively. These estimates are then combined by simply taking the ratio $J(r) = (1 - G(r))/(1 - F(r))$.

In fact several different estimates are computed using different edge corrections (Baddeley, 1998).

The Kaplan-Meier estimate (returned as `km`) is the ratio `J = (1-G)/(1-F)` of the Kaplan-Meier estimates of $1 - F$ and $1 - G$ computed by `Fest` and `Gest` respectively. This is computed if `correction=NULL` or if `correction` includes `"km"`.

The Hanisch-style estimate (returned as `han`) is the ratio `J = (1-G)/(1-F)` where `F` is the Chiu-Stoyan estimate of F and `G` is the Hanisch estimate of G. This is computed if `correction=NULL` or if `correction` includes `"cs"` or `"han"`.

The reduced-sample or border corrected estimate (returned as `rs`) is the same ratio `J = (1-G)/(1-F)` of the border corrected estimates. This is computed if `correction=NULL` or if `correction` includes `"rs"` or `"border"`.

These edge-corrected estimators are slightly biased for J, since they are ratios of approximately unbiased estimators. The logarithm of the Kaplan-Meier estimate is exactly unbiased for $\log J$.

The uncorrected estimate (returned as `un` and computed only if `correction` includes `"none"`) is the ratio `J = (1-G)/(1-F)` of the uncorrected ("raw") estimates of the survival functions of F and G, which are the empirical distribution functions of the empty space distances `Fest(X,...)$raw` and of the nearest neighbour distances `Gest(X,...)$raw`. The uncorrected estimates of F and G are severely biased. However the uncorrected estimate of J is approximately unbiased (if the process is close to Poisson); it is insensitive to edge effects, and should be used when edge effects are severe (see Baddeley et al, 2000).

The algorithm for `Fest` uses two discrete approximations which are controlled by the parameter `eps` and by the spacing of values of `r` respectively. See `Fest` for details. First-time users are strongly advised not to specify these arguments.

Note that the value returned by `Jest` includes the output of `Fest` and `Gest` as attributes (see the last example below). If the user is intending to compute the `F,G` and `J` functions for the point pattern, it is only necessary to call `Jest`.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing

`r`	the vector of values of the argument r at which the function J has been estimated
`rs`	the "reduced sample" or "border correction" estimator of $J(r)$ computed from the border-corrected estimates of F and G
`km`	the spatial Kaplan-Meier estimator of $J(r)$ computed from the Kaplan-Meier estimates of F and G
`han`	the Hanisch-style estimator of $J(r)$ computed from the Hanisch estimate of G and the Chiu-Stoyan estimate of F
`un`	the uncorrected estimate of $J(r)$ computed from the uncorrected estimates of F and G
`theo`	the theoretical value of $J(r)$ for a stationary Poisson process: identically equal to 1

The data frame also has **attributes**

`F`	the output of `Fest` for this point pattern, containing three estimates of the empty space function $F(r)$ and an estimate of its hazard function

G the output of `Gest` for this point pattern, containing three estimates of the nearest neighbour distance distribution function $G(r)$ and an estimate of its hazard function

Note

Sizeable amounts of memory may be needed during the calculation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37–78.

Baddeley, A.J. and Gill, R.D. The empty space hazard of a spatial pattern. Research Report 1994/3, Department of Mathematics, University of Western Australia, May 1994.

Baddeley, A.J. and Gill, R.D. Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25** (1997) 263–292.

Baddeley, A., Kerscher, M., Schladitz, K. and Scott, B.T. Estimating the *J* function without edge correction. *Statistica Neerlandica* **54** (2000) 315–328.

Borgefors, G. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34** (1986) 344–371.

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

Thonnes, E. and Van Lieshout, M.N.M, A comparative study on the power of Van Lieshout and Baddeley's J-function. *Biometrical Journal* **41** (1999) 721–734.

Van Lieshout, M.N.M. and Baddeley, A.J. A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50** (1996) 344–361.

See Also

`Jinhom`, `Fest`, `Gest`, `Kest`, `km.rs`, `reduced.sample`, `kaplan.meier`

Examples

```
data(cells)
J <- Jest(cells, 0.01)
plot(J, main="cells data")
# values are far above J = 1, indicating regular pattern

data(redwood)
J <- Jest(redwood, 0.01, legendpos="center")
plot(J, main="redwood data")
# values are below J = 1, indicating clustered pattern
```

Jinhom *Inhomogeneous J-function*

Description

Estimates the inhomogeneous J function of a non-stationary point pattern.

Usage

```
Jinhom(X, lambda = NULL, lmin = NULL, ...,
       sigma = NULL, varcov = NULL,
       r = NULL, breaks = NULL, update = TRUE)
```

Arguments

X	The observed data point pattern, from which an estimate of the inhomogeneous J function will be computed. An object of class "ppp" or in a format recognised by as.ppp()
lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm" or "kppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
lmin	Optional. The minimum possible value of the intensity over the spatial domain. A positive numerical value.
sigma,varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.
...	Extra arguments passed to as.mask to control the pixel resolution, or passed to density.ppp to control the smoothing bandwidth.
r	vector of values for the argument r at which the inhomogeneous K function should be evaluated. Not normally given by the user; there is a sensible default.
breaks	This argument is for internal use only.
update	Logical. If lambda is a fitted model (class "ppm" or "kppm") and update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without fitting it to X.

Details

This command computes estimates of the inhomogeneous J-function (Van Lieshout, 2010) of a point pattern. It is the counterpart, for inhomogeneous spatial point patterns, of the J function for homogeneous point patterns computed by Jest.

The argument X should be a point pattern (object of class "ppp").

The inhomogeneous J function is computed as $Jinhom(r) = (1 - Ginhom(r))/(1 - Finhom(r))$ where $Ginhom, Finhom$ are the inhomogeneous G and F functions computed using the border correction (equations (7) and (6) respectively in Van Lieshout, 2010).

The argument lambda should supply the (estimated) values of the intensity function λ of the point process. It may be either

a numeric vector containing the values of the intensity function at the points of the pattern X.

a pixel image (object of class "im") assumed to contain the values of the intensity function at all locations in the window.

a fitted point process model (object of class "ppm" or "kppm") whose fitted *trend* can be used as the fitted intensity. (If update=TRUE the model will first be refitted to the data X before the trend is computed.)

a function which can be evaluated to give values of the intensity at any locations.

omitted: if lambda is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

If lambda is a numeric vector, then its length should be equal to the number of points in the pattern X. The value lambda[i] is assumed to be the the (estimated) value of the intensity $\lambda(x_i)$ for the point x_i of the pattern X. Each value must be a positive number; NA's are not allowed.

If lambda is a pixel image, the domain of the image should cover the entire window of the point pattern. If it does not (which may occur near the boundary because of discretisation error), then the missing pixel values will be obtained by applying a Gaussian blur to lambda using blur, then looking up the values of this blurred image for the missing locations. (A warning will be issued in this case.)

If lambda is a function, then it will be evaluated in the form lambda(x,y) where x and y are vectors of coordinates of the points of X. It should return a numeric vector with length equal to the number of points in X.

If lambda is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate lambda[i] for the point X[i] is computed by removing X[i] from the point pattern, applying kernel smoothing to the remaining points using density.ppp, and evaluating the smoothed intensity at the point X[i]. The smoothing kernel bandwidth is controlled by the arguments sigma and varcov, which are passed to density.ppp along with any extra arguments.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Author(s)

Original code by Marie-Colette van Lieshout. C implementation and R adaptation by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Møller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.

van Lieshout, M.N.M. (2010) A J-function for inhomogeneous point processes. *Statistica Neerlandica* **65**, 183–201.

See Also

Ginhom, Finhom, Jest

Examples

```
## Not run:
  plot(Jinhom(swedishpines, sigma=bw.diggle, adjust=2))

## End(Not run)
  plot(Jinhom(swedishpines, sigma=10))
```

Jmulti *Marked J Function*

Description

For a marked point pattern, estimate the multitype J function summarising dependence between the points in subset I and those in subset J.

Usage

```
Jmulti(X, I, J, eps=NULL, r=NULL, breaks=NULL, ..., disjoint=NULL,
       correction=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of the multitype distance distribution function $J_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
I	Subset of points of X from which distances are measured. See Details.
J	Subset of points in X to which distances are measured. See Details.
eps	A positive number. The pixel resolution of the discrete approximation to Euclidean distance (see Jest). There is a sensible default.
r	numeric vector. The values of the argument r at which the distribution function $J_{IJ}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
...	Ignored.
disjoint	Optional flag indicating whether the subsets I and J are disjoint. If missing, this value will be computed by inspecting the vectors I and J.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "Hanisch" and "best". Alternatively correction="all" selects all options.

Details

The function Jmulti generalises Jest (for unmarked point patterns) and Jdot and Jcross (for multitype point patterns) to arbitrary marked point patterns.

Suppose X_I, X_J are subsets, possibly overlapping, of a marked point process. Define

$$J_{IJ}(r) = \frac{1 - G_{IJ}(r)}{1 - F_J(r)}$$

where $F_J(r)$ is the cumulative distribution function of the distance from a fixed location to the nearest point of X_J, and $G_{IJ}(r)$ is the distribution function of the distance from a typical point of X_I to the nearest distinct point of X_J.

The argument `X` must be a point pattern (object of class `"ppp"`) or any data that are acceptable to `as.ppp`.

The arguments `I` and `J` specify two subsets of the point pattern. They may be any type of subset indices, for example, logical vectors of length equal to `npoints(X)`, or integer vectors with entries in the range 1 to `npoints(X)`, or negative integer vectors.

Alternatively, `I` and `J` may be **functions** that will be applied to the point pattern `X` to obtain index vectors. If `I` is a function, then evaluating `I(X)` should yield a valid subset index. This option is useful when generating simulation envelopes using `envelope`.

It is assumed that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in `Jest`.

The argument `r` is the vector of values for the distance r at which $J_{IJ}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of `hist`) for the computation of histograms of distances. The reduced-sample and Kaplan-Meier estimators are computed from histogram counts. In the case of the Kaplan-Meier estimator this introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify `r`. However, if it is specified, `r` must satisfy `r[1] = 0`, and `max(r)` must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of `r` must be finely spaced.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing six numeric columns

`r`	the values of the argument r at which the function $J_{IJ}(r)$ has been estimated
`rs`	the "reduced sample" or "border correction" estimator of $J_{IJ}(r)$
`km`	the spatial Kaplan-Meier estimator of $J_{IJ}(r)$
`han`	the Hanisch-style estimator of $J_{IJ}(r)$
`un`	the uncorrected estimate of $J_{IJ}(r)$, formed by taking the ratio of uncorrected empirical estimators of $1 - G_{IJ}(r)$ and $1 - F_J(r)$, see `Gdot` and `Fest`.
`theo`	the theoretical value of $J_{IJ}(r)$ for a marked Poisson process with the same estimated intensity, namely 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

`Jcross`, `Jdot`, `Jest`

Examples

```
trees <- longleaf
 # Longleaf Pine data: marks represent diameter

Jm <- Jmulti(trees, marks(trees) <= 15, marks(trees) >= 25)
plot(Jm)
```

K3est *K-function of a Three-Dimensional Point Pattern*

Description

Estimates the K-function from a three-dimensional point pattern.

Usage

```
K3est(X, ...,
      rmax = NULL, nrval = 128,
      correction = c("translation", "isotropic"),
      ratio=FALSE)
```

Arguments

X	Three-dimensional point pattern (object of class "pp3").
...	Ignored.
rmax	Optional. Maximum value of argument r for which $K_3(r)$ will be estimated.
nrval	Optional. Number of values of r for which $K_3(r)$ will be estimated. A large value of nrval is required to avoid discretisation effects.
correction	Optional. Character vector specifying the edge correction(s) to be applied. See Details.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

For a stationary point process Φ in three-dimensional space, the three-dimensional K function is

$$K_3(r) = \frac{1}{\lambda} E(N(\Phi, x, r) \mid x \in \Phi)$$

where λ is the intensity of the process (the expected number of points per unit volume) and $N(\Phi, x, r)$ is the number of points of Φ, other than x itself, which fall within a distance r of x. This is the three-dimensional generalisation of Ripley's K function for two-dimensional point processes (Ripley, 1977).

The three-dimensional point pattern X is assumed to be a partial realisation of a stationary point process Φ. The distance between each pair of distinct points is computed. The empirical cumulative distribution function of these values, with appropriate edge corrections, is renormalised to give the estimate of $K_3(r)$.

The available edge corrections are:

"translation": the Ohser translation correction estimator (Ohser, 1983; Baddeley et al, 1993)

"isotropic": the three-dimensional counterpart of Ripley's isotropic edge correction (Ripley, 1977; Baddeley et al, 1993).

Alternatively correction="all" selects all options.

Value

A function value table (object of class "fv") that can be plotted, printed or coerced to a data frame containing the function values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rana Moyeed.

References

Baddeley, A.J, Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.

Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

F3est, G3est, pcf3est

Examples

```
X <- rpoispp3(42)
Z <- K3est(X)
if(interactive()) plot(Z)
```

---

kaplan.meier *Kaplan-Meier Estimator using Histogram Data*

Description

Compute the Kaplan-Meier estimator of a survival time distribution function, from histogram data

Usage

```
kaplan.meier(obs, nco, breaks, upperobs=0)
```

Arguments

obs	vector of n integers giving the histogram of all observations (censored or uncensored survival times)
nco	vector of n integers giving the histogram of uncensored observations (those survival times that are less than or equal to the censoring time)
breaks	Vector of $n + 1$ breakpoints which were used to form both histograms.
upperobs	Number of observations beyond the rightmost breakpoint, if any.

Details

This function is needed mainly for internal use in **spatstat**, but may be useful in other applications where you want to form the Kaplan-Meier estimator from a huge dataset.

Suppose T_i are the survival times of individuals $i = 1, \ldots, M$ with unknown distribution function $F(t)$ which we wish to estimate. Suppose these times are right-censored by random censoring times C_i. Thus the observations consist of right-censored survival times $\tilde{T}_i = \min(T_i, C_i)$ and non-censoring indicators $D_i = 1\{T_i \leq C_i\}$ for each i.

If the number of observations M is large, it is efficient to use histograms. Form the histogram `obs` of all observed times $\tilde{T}_i$. That is, `obs[k]` counts the number of values $\tilde{T}_i$ in the interval `(breaks[k],breaks[k+1]]` for $k > 1$ and `[breaks[1],breaks[2]]` for $k = 1$. Also form the histogram `nco` of all uncensored times, i.e. those $\tilde{T}_i$ such that $D_i = 1$. These two histograms are the arguments passed to `kaplan.meier`.

The vectors `km` and `lambda` returned by `kaplan.meier` are (histogram approximations to) the Kaplan-Meier estimator of $F(t)$ and its hazard rate $\lambda(t)$. Specifically, `km[k]` is an estimate of `F(breaks[k+1])`, and `lambda[k]` is an estimate of the average of $\lambda(t)$ over the interval `(breaks[k],breaks[k+1])`.

The histogram breaks must include 0. If the histogram breaks do not span the range of the observations, it is important to count how many survival times $\tilde{T}_i$ exceed the rightmost breakpoint, and give this as the value `upperobs`.

Value

A list with two elements:

km	Kaplan-Meier estimate of the survival time c.d.f. $F(t)$
lambda	corresponding Nelson-Aalen estimate of the hazard rate $\lambda(t)$

These are numeric vectors of length n.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`reduced.sample`, `km.rs`

Kcom	*Model Compensator of K Function*

Description

Given a point process model fitted to a point pattern dataset, this function computes the *compensator* of the K function based on the fitted model (as well as the usual nonparametric estimates of K based on the data alone). Comparison between the nonparametric and model-compensated K functions serves as a diagnostic for the model.

Usage

```
Kcom(object, r = NULL, breaks = NULL, ...,
     correction = c("border", "isotropic", "translate"),
     conditional = !is.poisson(object),
     restrict = FALSE,
     model = NULL,
     trend = ~1, interaction = Poisson(), rbord = reach(interaction),
     compute.var = TRUE,
     truecoef = NULL, hi.res = NULL)
```

Arguments

`object`	Object to be analysed. Either a fitted point process model (object of class "ppm") or a point pattern (object of class "ppp") or quadrature scheme (object of class "quad").
`r`	Optional. Vector of values of the argument r at which the function $K(r)$ should be computed. This argument is usually not specified. There is a sensible default.
`breaks`	This argument is for advanced use only.
`...`	Ignored.
`correction`	Optional vector of character strings specifying the edge correction(s) to be used. See `Kest` for options.
`conditional`	Optional. Logical value indicating whether to compute the estimates for the conditional case. See Details.
`restrict`	Logical value indicating whether to compute the restriction estimator (`restrict=TRUE`) or the reweighting estimator (`restrict=FALSE`, the default). Applies only if `conditional=TRUE`. See Details.
`model`	Optional. A fitted point process model (object of class "ppm") to be re-fitted to the data using `update.ppm`, if `object` is a point pattern. Overrides the arguments `trend,interaction,rbord`.
`trend,interaction,rbord`	Optional. Arguments passed to `ppm` to fit a point process model to the data, if `object` is a point pattern. See `ppm` for details.
`compute.var`	Logical value indicating whether to compute the Poincare variance bound for the residual K function (calculation is only implemented for the isotropic correction).
`truecoef`	Optional. Numeric vector. If present, this will be treated as if it were the true coefficient vector of the point process model, in calculating the diagnostic. Incompatible with `hi.res`.

hi.res Optional. List of parameters passed to quadscheme. If this argument is present, the model will be re-fitted at high resolution as specified by these parameters. The coefficients of the resulting fitted model will be taken as the true coefficients. Then the diagnostic will be computed for the default quadrature scheme, but using the high resolution coefficients.

Details

This command provides a diagnostic for the goodness-of-fit of a point process model fitted to a point pattern dataset. It computes an estimate of the K function of the dataset, together with a *model compensator* of the K function, which should be approximately equal if the model is a good fit to the data.

The first argument, `object`, is usually a fitted point process model (object of class `"ppm"`), obtained from the model-fitting function `ppm`.

For convenience, `object` can also be a point pattern (object of class `"ppp"`). In that case, a point process model will be fitted to it, by calling `ppm` using the arguments `trend` (for the first order trend), `interaction` (for the interpoint interaction) and `rbord` (for the erosion distance in the border correction for the pseudolikelihood). See `ppm` for details of these arguments.

The algorithm first extracts the original point pattern dataset (to which the model was fitted) and computes the standard nonparametric estimates of the K function. It then also computes the *model compensator* of the K function. The different function estimates are returned as columns in a data frame (of class `"fv"`).

The argument `correction` determines the edge correction(s) to be applied. See `Kest` for explanation of the principle of edge corrections. The following table gives the options for the `correction` argument, and the corresponding column names in the result:

`correction`	**description of correction**	**nonparametric**	**compensator**
`"isotropic"`	Ripley isotropic correction	`iso`	`icom`
`"translate"`	Ohser-Stoyan translation correction	`trans`	`tcom`
`"border"`	border correction	`border`	`bcom`

The nonparametric estimates can all be expressed in the form

$$\hat{K}(r) = \sum_i \sum_{j<i} e(x_i, x_j, r, x) I\{d(x_i, x_j) \le r\}$$

where x_i is the i-th data point, $d(x_i, x_j)$ is the distance between x_i and x_j, and $e(x_i, x_j, r, x)$ is a term that serves to correct edge effects and to re-normalise the sum. The corresponding model compensator is

$$\mathbf{C}\,\tilde{K}(r) = \int_W \lambda(u, x) \sum_j e(u, x_j, r, x \cup u) I\{d(u, x_j) \le r\}$$

where the integral is over all locations u in the observation window, $\lambda(u, x)$ denotes the conditional intensity of the model at the location u, and $x \cup u$ denotes the data point pattern x augmented by adding the extra point u.

If the fitted model is a Poisson point process, then the formulae above are exactly what is computed. If the fitted model is not Poisson, the formulae above are modified slightly to handle edge effects.

The modification is determined by the arguments `conditional` and `restrict`. The value of `conditional` defaults to `FALSE` for Poisson models and `TRUE` for non-Poisson models. If `conditional=FALSE` then the formulae above are not modified. If `conditional=TRUE`, then the algorithm calculates the *restriction estimator* if `restrict=TRUE`, and calculates the *reweighting estimator* if `restrict=FALSE`. See Appendix D of Baddeley, Rubak and Møller (2011). Thus, by default, the reweighting estimator is computed for non-Poisson models.

The nonparametric estimates of $K(r)$ are approximately unbiased estimates of the K-function, assuming the point process is stationary. The model compensators are unbiased estimates *of the mean values of the corresponding nonparametric estimates*, assuming the model is true. Thus, if the model is a good fit, the mean value of the difference between the nonparametric estimates and model compensators is approximately zero.

Value

A function value table (object of class `"fv"`), essentially a data frame of function values. There is a plot method for this class. See `fv.object`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

Ege Rubak `<rubak@math.aau.dk>` and Jesper Møller.

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Related functions: `Kres`, `Kest`.

Alternative functions: `Gcom`, `psstG`, `psstA`, `psst`.

Point process models: `ppm`.

Examples

```
    fit0 <- ppm(cells, ~1) # uniform Poisson


    if(interactive()) {
      plot(Kcom(fit0))
# compare the isotropic-correction estimates
      plot(Kcom(fit0), cbind(iso, icom) ~ r)
# uniform Poisson is clearly not correct
    }

    fit1 <- ppm(cells, ~1, Strauss(0.08))

    K1 <- Kcom(fit1)
    K1
    if(interactive()) {
      plot(K1)
      plot(K1, cbind(iso, icom) ~ r)
      plot(K1, cbind(trans, tcom) ~ r)
# how to plot the difference between nonparametric estimates and compensators
      plot(K1, iso - icom ~ r)
# fit looks approximately OK; try adjusting interaction distance
    }
```

```
fit2 <- ppm(cells, ~1, Strauss(0.12))

K2 <- Kcom(fit2)
if(interactive()) {
  plot(K2)
  plot(K2, cbind(iso, icom) ~ r)
  plot(K2, iso - icom ~ r)
}
```

Kcross *Multitype K Function (Cross-type)*

Description

For a multitype point pattern, estimate the multitype K function which counts the expected number of points of type j within a given distance of a point of type i.

Usage

```
Kcross(X, i, j, r=NULL, breaks=NULL, correction,
       ..., ratio=FALSE, from, to )
```

Arguments

X	The observed point pattern, from which an estimate of the cross type K function $K_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
r	numeric vector. The values of the argument r at which the distribution function $K_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
...	Ignored.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
from,to	An alternative way to specify i and j respectively.

Details

This function `Kcross` and its companions `Kdot` and `Kmulti` are generalisations of the function `Kest` to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible "colours" or "types". In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument `X` must be a point pattern (object of class `"ppp"`) or any data that are acceptable to `as.ppp`. It must be a marked point pattern, and the mark vector `X$marks` must be a factor.

The arguments `i` and `j` will be interpreted as levels of the factor `X$marks`. If `i` and `j` are missing, they default to the first and second level of the marks factor, respectively.

The "cross-type" (type i to type j) K function of a stationary multitype point process X is defined so that $\lambda_j K_{ij}(r)$ equals the expected number of additional random points of type j within a distance r of a typical point of type i in the process X. Here λ_j is the intensity of the type j points, i.e. the expected number of points of type j per unit area. The function K_{ij} is determined by the second order moment properties of X.

An estimate of $K_{ij}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the process of type i points were independent of the process of type j points, then $K_{ij}(r)$ would equal πr^2. Deviations between the empirical K_{ij} curve and the theoretical curve πr^2 may suggest dependence between the points of types i and j.

This algorithm estimates the distribution function $K_{ij}(r)$ from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in `Kest`, using the border correction.

The argument `r` is the vector of values for the distance r at which $K_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

The pair correlation function can also be applied to the result of `Kcross`; see `pcf`.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing numeric columns

`r`	the values of the argument r at which the function $K_{ij}(r)$ has been estimated
`theo`	the theoretical value of $K_{ij}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $K_{ij}(r)$ obtained by the edge corrections named.

If `ratio=TRUE` then the return value also has two attributes called `"numerator"` and `"denominator"` which are `"fv"` objects containing the numerators and denominators of each estimate of $K(r)$.

Warnings

The arguments `i` and `j` are always interpreted as levels of the factor `X$marks`. They are converted to character strings if they are not already character strings. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

Kdot, Kest, Kmulti, pcf

Examples

```
    # amacrine cells data
    K01 <- Kcross(amacrine, "off", "on")
    plot(K01)


    ## Not run:
    K10 <- Kcross(amacrine, "on", "off")

    # synthetic example: point pattern with marks 0 and 1
    pp <- runifpoispp(50)
    pp <- pp %mark% factor(sample(0:1, npoints(pp), replace=TRUE))
    K <- Kcross(pp, "0", "1")
    K <- Kcross(pp, 0, 1) # equivalent

## End(Not run)
```

Kcross.inhom *Inhomogeneous Cross K Function*

Description

For a multitype point pattern, estimate the inhomogeneous version of the cross K function, which counts the expected number of points of type j within a given distance of a point of type i, adjusted for spatially varying intensity.

Usage

```
Kcross.inhom(X, i, j, lambdaI=NULL, lambdaJ=NULL, ...,  r=NULL, breaks=NULL,
         correction = c("border", "isotropic", "Ripley", "translate"),
         sigma=NULL, varcov=NULL,
         lambdaIJ=NULL,
         lambdaX=NULL, update=TRUE, leaveoneout=TRUE)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous cross type K function $K_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
lambdaI	Optional. Values of the the estimated intensity of the sub-process of points of type i. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the type i points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdaJ	Optional. Values of the the estimated intensity of the sub-process of points of type j. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the type j points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
r	Optional. Numeric vector giving the values of the argument r at which the cross K function $K_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for advanced use only.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley" ,"translate", "translation", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
...	Ignored.
sigma	Standard deviation of isotropic Gaussian smoothing kernel, used in computing leave-one-out kernel estimates of lambdaI, lambdaJ if they are omitted.
varcov	Variance-covariance matrix of anisotropic Gaussian kernel, used in computing leave-one-out kernel estimates of lambdaI, lambdaJ if they are omitted. Incompatible with sigma.
lambdaIJ	Optional. A matrix containing estimates of the product of the intensities lambdaI and lambdaJ for each pair of points of types i and j respectively.
lambdaX	Optional. Values of the intensity for all points of X. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location. If present, this argument overrides both lambdaI and lambdaJ.

update — Logical value indicating what to do when `lambdaI`, `lambdaJ` or `lambdaX` is a fitted point process model (class "ppm", "kppm" or "dppm"). If `update=TRUE` (the default), the model will first be refitted to the data `X` (using `update.ppm` or `update.kppm`) before the fitted intensity is computed. If `update=FALSE`, the fitted intensity of the model will be computed without re-fitting it to `X`.

leaveoneout — Logical value (passed to `density.ppp` or `fitted.ppm`) specifying whether to use a leave-one-out rule when calculating the intensity.

Details

This is a generalisation of the function `Kcross` to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function `Kinhom`.

The inhomogeneous cross-type K function is described by Møller and Waagepetersen (2003, pages 48-49 and 51-53).

Briefly, given a multitype point process, suppose the sub-process of points of type j has intensity function $\lambda_j(u)$ at spatial locations u. Suppose we place a mass of $1/\lambda_j(\zeta)$ at each point ζ of type j. Then the expected total mass per unit area is 1. The inhomogeneous "cross-type" K function $K_{ij}^{\mathrm{inhom}}(r)$ equals the expected total mass within a radius r of a point of the process of type i.

If the process of type i points were independent of the process of type j points, then $K_{ij}^{\mathrm{inhom}}(r)$ would equal πr^2. Deviations between the empirical K_{ij} curve and the theoretical curve πr^2 suggest dependence between the points of types i and j.

The argument `X` must be a point pattern (object of class "ppp") or any data that are acceptable to `as.ppp`. It must be a marked point pattern, and the mark vector `X$marks` must be a factor.

The arguments `i` and `j` will be interpreted as levels of the factor `X$marks`. (Warning: this means that an integer value `i=3` will be interpreted as the number 3, **not** the 3rd smallest level). If `i` and `j` are missing, they default to the first and second level of the marks factor, respectively.

The argument `lambdaI` supplies the values of the intensity of the sub-process of points of type `i`. It may be either

a pixel image (object of class "im") which gives the values of the type `i` intensity at all locations in the window containing `X`;

a numeric vector containing the values of the type `i` intensity evaluated only at the data points of type `i`. The length of this vector must equal the number of type `i` points in `X`.

a function which can be evaluated to give values of the intensity at any locations.

a fitted point process model (object of class "ppm", "kppm" or "dppm") whose fitted *trend* can be used as the fitted intensity. (If `update=TRUE` the model will first be refitted to the data `X` before the trend is computed.)

omitted: if `lambdaI` is omitted then it will be estimated using a leave-one-out kernel smoother.

If `lambdaI` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate of `lambdaI` for a given point is computed by removing the point from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point in question. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Similarly `lambdaJ` should contain estimated values of the intensity of the sub-process of points of type `j`. It may be either a pixel image, a function, a numeric vector, or omitted.

Alternatively if the argument `lambdaX` is given, then it specifies the intensity values for all points of `X`, and the arguments `lambdaI`, `lambdaJ` will be ignored.

The optional argument lambdaIJ is for advanced use only. It is a matrix containing estimated values of the products of these two intensities for each pair of data points of types i and j respectively.

The argument r is the vector of values for the distance r at which $K_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must exceed the radius of the largest disc contained in the window.

The argument correction chooses the edge correction as explained e.g. in Kest.

The pair correlation function can also be applied to the result of Kcross.inhom; see pcf.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $K_{ij}(r)$ has been estimated
theo	the theoretical value of $K_{ij}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{ij}(r)$ obtained by the edge corrections named.

Warnings

The arguments i and j are always interpreted as levels of the factor X$marks. They are converted to character strings if they are not already character strings. The value i=1 does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Møller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

Møller, J. and Waagepetersen, R. Statistical Inference and Simulation for Spatial Point Processes Chapman and Hall/CRC Boca Raton, 2003.

See Also

Kcross, Kinhom, Kdot.inhom, Kmulti.inhom, pcf

Examples

```
# Lansing Woods data
woods <- lansing

ma <- split(woods)$maple
wh <- split(woods)$whiteoak

# method (1): estimate intensities by nonparametric smoothing
lambdaM <- density.ppp(ma, sigma=0.15, at="points")
lambdaW <- density.ppp(wh, sigma=0.15, at="points")
```

```
K <- Kcross.inhom(woods, "whiteoak", "maple", lambdaW, lambdaM)

# method (2): leave-one-out
K <- Kcross.inhom(woods, "whiteoak", "maple", sigma=0.15)

# method (3): fit parametric intensity model
fit <- ppm(woods ~marks * polynom(x,y,2))
# alternative (a): use fitted model as 'lambda' argument
K <- Kcross.inhom(woods, "whiteoak", "maple",
                  lambdaI=fit, lambdaJ=fit, update=FALSE)
K <- Kcross.inhom(woods, "whiteoak", "maple",
                  lambdaX=fit, update=FALSE)
# alternative (b): evaluate fitted intensities at data points
# (these are the intensities of the sub-processes of each type)
inten <- fitted(fit, dataonly=TRUE)
# split according to types of points
lambda <- split(inten, marks(woods))
K <- Kcross.inhom(woods, "whiteoak", "maple",
          lambda$whiteoak, lambda$maple)

# synthetic example: type A points have intensity 50,
#                    type B points have intensity 100 * x
lamB <- as.im(function(x,y){50 + 100 * x}, owin())
X <- superimpose(A=runifpoispp(50), B=rpoispp(lamB))
K <- Kcross.inhom(X, "A", "B",
    lambdaI=as.im(50, Window(X)), lambdaJ=lamB)
```

Kdot *Multitype K Function (i-to-any)*

Description

For a multitype point pattern, estimate the multitype K function which counts the expected number of other points of the process within a given distance of a point of type i.

Usage

```
Kdot(X, i, r=NULL, breaks=NULL, correction, ..., ratio=FALSE, from)
```

Arguments

X	The observed point pattern, from which an estimate of the multitype K function $K_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
r	numeric vector. The values of the argument r at which the distribution function $K_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.

correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
...	Ignored.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
from	An alternative way to specify i.

Details

This function Kdot and its companions Kcross and Kmulti are generalisations of the function Kest to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible "colours" or "types". In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor.

The argument i will be interpreted as a level of the factor X$marks. If i is missing, it defaults to the first level of the marks factor, i = levels(X$marks)[1].

The "type i to any type" multitype K function of a stationary multitype point process X is defined so that $\lambda K_{i\bullet}(r)$ equals the expected number of additional random points within a distance r of a typical point of type i in the process X. Here λ is the intensity of the process, i.e. the expected number of points of X per unit area. The function $K_{i\bullet}$ is determined by the second order moment properties of X.

An estimate of $K_{i\bullet}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the subprocess of type i points were independent of the subprocess of points of all types not equal to i, then $K_{i\bullet}(r)$ would equal πr^2. Deviations between the empirical $K_{i\bullet}$ curve and the theoretical curve πr^2 may suggest dependence between types.

This algorithm estimates the distribution function $K_{i\bullet}(r)$ from the point pattern X. It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in Kest, using the border correction.

The argument r is the vector of values for the distance r at which $K_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must exceed the radius of the largest disc contained in the window.

The pair correlation function can also be applied to the result of Kdot; see pcf.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $K_{i\bullet}(r)$ has been estimated
theo	the theoretical value of $K_{i\bullet}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{i\bullet}(r)$ obtained by the edge corrections named.

If ratio=TRUE then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $K(r)$.

Warnings

The argument i is interpreted as a level of the factor X$marks. It is converted to a character string if it is not already a character string. The value i=1 does **not** refer to the first level of the factor.

The reduced sample estimator of $K_{i\bullet}$ is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r. Its range is always within $[0, 1]$.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

Kdot, Kest, Kmulti, pcf

Examples

```
 # Lansing woods data: 6 types of trees
woods <- lansing



Kh. <- Kdot(woods, "hickory")
# diagnostic plot for independence between hickories and other trees
plot(Kh.)

## Not run:
# synthetic example with two marks "a" and "b"
pp <- runifpoispp(50)
pp <- pp %mark% factor(sample(c("a","b"), npoints(pp), replace=TRUE))
K <- Kdot(pp, "a")

## End(Not run)
```

Kdot.inhom *Inhomogeneous Multitype K Dot Function*

Description

For a multitype point pattern, estimate the inhomogeneous version of the dot K function, which counts the expected number of points of any type within a given distance of a point of type i, adjusted for spatially varying intensity.

Usage

```
Kdot.inhom(X, i, lambdaI=NULL, lambdadot=NULL, ..., r=NULL, breaks=NULL,
         correction = c("border", "isotropic", "Ripley", "translate"),
         sigma=NULL, varcov=NULL, lambdaIdot=NULL,
         lambdaX=NULL, update=TRUE, leaveoneout=TRUE)
```

Arguments

`X`	The observed point pattern, from which an estimate of the inhomogeneous cross type K function $K_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
`i`	The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.
`lambdaI`	Optional. Values of the estimated intensity of the sub-process of points of type `i`. Either a pixel image (object of class `"im"`), a numeric vector containing the intensity values at each of the type `i` points in `X`, a fitted point process model (object of class `"ppm"` or `"kppm"` or `"dppm"`), or a `function(x,y)` which can be evaluated to give the intensity value at any location.
`lambdadot`	Optional. Values of the estimated intensity of the entire point process, Either a pixel image (object of class `"im"`), a numeric vector containing the intensity values at each of the points in `X`, a fitted point process model (object of class `"ppm"` or `"kppm"` or `"dppm"`), or a `function(x,y)` which can be evaluated to give the intensity value at any location.
`...`	Ignored.
`r`	Optional. Numeric vector giving the values of the argument r at which the cross K function $K_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`breaks`	This argument is for internal use only.
`correction`	A character vector containing any selection of the options `"border"`, `"bord.modif"`, `"isotropic"`, `"Ripley"`, `"translate"`, `"translation"`, `"none"` or `"best"`. It specifies the edge correction(s) to be applied. Alternatively `correction="all"` selects all options.
`sigma`	Standard deviation of isotropic Gaussian smoothing kernel, used in computing leave-one-out kernel estimates of `lambdaI`, `lambdadot` if they are omitted.
`varcov`	Variance-covariance matrix of anisotropic Gaussian kernel, used in computing leave-one-out kernel estimates of `lambdaI`, `lambdadot` if they are omitted. Incompatible with `sigma`.

lambdaIdot
: Optional. A matrix containing estimates of the product of the intensities lambdaI and lambdadot for each pair of points, the first point of type i and the second of any type.

lambdaX
: Optional. Values of the intensity for all points of X. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location. If present, this argument overrides both lambdaI and lambdadot.

update
: Logical value indicating what to do when lambdaI, lambdadot or lambdaX is a fitted point process model (class "ppm", "kppm" or "dppm"). If update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without re-fitting it to X.

leaveoneout
: Logical value (passed to density.ppp or fitted.ppm) specifying whether to use a leave-one-out rule when calculating the intensity.

Details

This is a generalisation of the function Kdot to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function Kinhom.

Briefly, given a multitype point process, consider the points without their types, and suppose this unmarked point process has intensity function $\lambda(u)$ at spatial locations u. Suppose we place a mass of $1/\lambda(\zeta)$ at each point ζ of the process. Then the expected total mass per unit area is 1. The inhomogeneous "dot-type" K function $K_{i\bullet}^{\mathrm{inhom}}(r)$ equals the expected total mass within a radius r of a point of the process of type i, discounting this point itself.

If the process of type i points were independent of the points of other types, then $K_{i\bullet}^{\mathrm{inhom}}(r)$ would equal πr^2. Deviations between the empirical $K_{i\bullet}$ curve and the theoretical curve πr^2 suggest dependence between the points of types i and j for $j \neq i$.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor.

The argument i will be interpreted as a level of the factor X$marks. (Warning: this means that an integer value i=3 will be interpreted as the number 3, **not** the 3rd smallest level). If i is missing, it defaults to the first level of the marks factor, i = levels(X$marks)[1].

The argument lambdaI supplies the values of the intensity of the sub-process of points of type i. It may be either

a pixel image (object of class "im") which gives the values of the type i intensity at all locations in the window containing X;

a numeric vector containing the values of the type i intensity evaluated only at the data points of type i. The length of this vector must equal the number of type i points in X.

a function of the form function(x,y) which can be evaluated to give values of the intensity at any locations.

a fitted point process model (object of class "ppm", "kppm" or "dppm") whose fitted *trend* can be used as the fitted intensity. (If update=TRUE the model will first be refitted to the data X before the trend is computed.)

omitted: if lambdaI is omitted then it will be estimated using a leave-one-out kernel smoother.

If `lambdaI` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate of `lambdaI` for a given point is computed by removing the point from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point in question. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Similarly the argument `lambdadot` should contain estimated values of the intensity of the entire point process. It may be either a pixel image, a numeric vector of length equal to the number of points in `X`, a function, or omitted.

Alternatively if the argument `lambdaX` is given, then it specifies the intensity values for all points of `X`, and the arguments `lambdaI`, `lambdadot` will be ignored. (The two arguments `lambdaI`, `lambdadot` allow the user to specify two different methods for calculating the intensities of the two kinds of points, while `lambdaX` ensures that the same method is used for both kinds of points.)

For advanced use only, the optional argument `lambdaIdot` is a matrix containing estimated values of the products of these two intensities for each pair of points, the first point of type `i` and the second of any type.

The argument `r` is the vector of values for the distance r at which $K_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must exceed the radius of the largest disc contained in the window.

The argument `correction` chooses the edge correction as explained e.g. in `Kest`.

The pair correlation function can also be applied to the result of `Kcross.inhom`; see `pcf`.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing numeric columns

`r`	the values of the argument r at which the function $K_{i\bullet}(r)$ has been estimated
`theo`	the theoretical value of $K_{i\bullet}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $K_{i\bullet}(r)$ obtained by the edge corrections named.

Warnings

The argument `i` is interpreted as a level of the factor `X$marks`. It is converted to a character string if it is not already a character string. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

References

Møller, J. and Waagepetersen, R. Statistical Inference and Simulation for Spatial Point Processes Chapman and Hall/CRC Boca Raton, 2003.

See Also

`Kdot`, `Kinhom`, `Kcross.inhom`, `Kmulti.inhom`, `pcf`

Examples

```
# Lansing Woods data
woods <- lansing
woods <- woods[seq(1,npoints(woods), by=10)]
ma <- split(woods)$maple
lg <- unmark(woods)

# Estimate intensities by nonparametric smoothing
lambdaM <- density.ppp(ma, sigma=0.15, at="points")
lambdadot <- density.ppp(lg, sigma=0.15, at="points")
K <- Kdot.inhom(woods, "maple", lambdaI=lambdaM,
                                  lambdadot=lambdadot)

# Equivalent
K <- Kdot.inhom(woods, "maple", sigma=0.15)

# Fit model
fit <- ppm(woods ~ marks * polynom(x,y,2))
K <- Kdot.inhom(woods, "maple", lambdaX=fit, update=FALSE)

# synthetic example: type A points have intensity 50,
#                    type B points have intensity 50 + 100 * x
lamB <- as.im(function(x,y){50 + 100 * x}, owin())
lamdot <- as.im(function(x,y) { 100 + 100 * x}, owin())
X <- superimpose(A=runifpoispp(50), B=rpoispp(lamB))
K <- Kdot.inhom(X, "B",  lambdaI=lamB,     lambdadot=lamdot)
```

`kernel.factor` *Scale factor for density kernel*

Description

Returns a scale factor for the kernels used in density estimation for numerical data.

Usage

```
kernel.factor(kernel = "gaussian")
```

Arguments

`kernel`	String name of the kernel. Options are `"gaussian"`, `"rectangular"`, `"triangular"`, `"epanechnikov"`, `"biweight"`, `"cosine"` and `"optcosine"`. (Partial matching is used).

Details

Kernel estimation of a probability density in one dimension is performed by `density.default` using a kernel function selected from the list above.

This function computes a scale constant for the kernel. For the Gaussian kernel, this constant is equal to 1. Otherwise, the constant c is such that the kernel with standard deviation 1 is supported on the interval $[-c, c]$.

For more information about these kernels, see `density.default`.

Value

A single number.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Martin Hazelton

See Also

density.default, dkernel, kernel.moment, kernel.squint

Examples

```
kernel.factor("rect")
# bandwidth for Epanechnikov kernel with half-width h=1
h <- 1
bw <- h/kernel.factor("epa")
```

---

kernel.moment *Moment of Smoothing Kernel*

Description

Computes the complete or incomplete mth moment of a smoothing kernel.

Usage

```
kernel.moment(m, r, kernel = "gaussian")
```

Arguments

m	Exponent (order of moment). An integer.
r	Upper limit of integration for the incomplete moment. A numeric value or numeric vector. Set r=Inf to obtain the complete moment.
kernel	String name of the kernel. Options are "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine" and "optcosine". (Partial matching is used).

Details

Kernel estimation of a probability density in one dimension is performed by density.default using a kernel function selected from the list above. For more information about these kernels, see density.default.

The function kernel.moment computes the partial integral

$$\int_{-\infty}^{r} t^m k(t) dt$$

where $k(t)$ is the selected kernel, r is the upper limit of integration, and m is the exponent or order.

Value

A single number, or a numeric vector of the same length as r.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Martin Hazelton.

See Also

density.default, dkernel, kernel.factor,

Examples

```
kernel.moment(1, 0.1, "epa")
curve(kernel.moment(2, x, "epa"), from=-1, to=1)
```

---

kernel.squint *Integral of Squared Kernel*

Description

Computes the integral of the squared kernel, for the kernels used in density estimation for numerical data.

Usage

```
kernel.squint(kernel = "gaussian", bw=1)
```

Arguments

kernel	String name of the kernel. Options are "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine" and "optcosine". (Partial matching is used).
bw	Bandwidth (standard deviation) of the kernel.

Details

Kernel estimation of a probability density in one dimension is performed by density.default using a kernel function selected from the list above.

This function computes the integral of the squared kernel,

$$R = \int_{-\infty}^{\infty} k(x)^2 \, \mathrm{d}x$$

where $k(x)$ is the kernel with bandwidth bw.

Value

A single number.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> and Martin Hazelton

See Also

density.default, dkernel, kernel.moment, kernel.factor

Examples

```
kernel.squint("gaussian", 3)

# integral of squared Epanechnikov kernel with half-width h=1
h <- 1
bw <- h/kernel.factor("epa")
kernel.squint("epa", bw)
```

Kest *K-function*

Description

Estimates Ripley's reduced second moment function $K(r)$ from a point pattern in a window of arbitrary shape.

Usage

```
Kest(X, ..., r=NULL, rmax=NULL, breaks=NULL,
   correction=c("border", "isotropic", "Ripley", "translate"),
  nlarge=3000, domain=NULL, var.approx=FALSE, ratio=FALSE)
```

Arguments

X	The observed point pattern, from which an estimate of $K(r)$ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp().
...	Ignored.
r	Optional. Vector of values for the argument r at which $K(r)$ should be evaluated. Users are advised *not* to specify this argument; there is a sensible default. If necessary, specify rmax.
rmax	Optional. Maximum desired value of the argument r.
breaks	This argument is for internal use only.
correction	Optional. A character vector containing any selection of the options "none", "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "rigid", "none", "good" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
nlarge	Optional. Efficiency threshold. If the number of points exceeds nlarge, then only the border correction will be computed (by default), using a fast algorithm.
domain	Optional. Calculations will be restricted to this subset of the window. See Details.

`var.approx` Logical. If `TRUE`, the approximate variance of $\hat{K}(r)$ under CSR will also be computed.

`ratio` Logical. If `TRUE`, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

The K function (variously called "Ripley's K-function" and the "reduced second moment function") of a stationary point process X is defined so that $\lambda K(r)$ equals the expected number of additional random points within a distance r of a typical random point of X. Here λ is the intensity of the process, i.e. the expected number of points of X per unit area. The K function is determined by the second order moment properties of X.

An estimate of K derived from a spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern (Cressie, 1991; Diggle, 1983; Ripley, 1977, 1988). In exploratory analyses, the estimate of K is a useful statistic summarising aspects of inter-point "dependence" and "clustering". For inferential purposes, the estimate of K is usually compared to the true value of K for a completely random (Poisson) point process, which is $K(r) = \pi r^2$. Deviations between the empirical and theoretical K curves may suggest spatial clustering or spatial regularity.

This routine `Kest` estimates the K function of a stationary point process, given observation of the process inside a known, bounded window. The argument `X` is interpreted as a point pattern object (of class `"ppp"`, see `ppp.object`) and can be supplied in any of the formats recognised by `as.ppp()`.

The estimation of K is hampered by edge effects arising from the unobservability of points of the random pattern outside the window. An edge correction is needed to reduce bias (Baddeley, 1998; Ripley, 1988). The corrections implemented here are

border the border method or "reduced sample" estimator (see Ripley, 1988). This is the least efficient (statistically) and the fastest to compute. It can be computed for a window of arbitrary shape.

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is implemented for rectangular and polygonal windows (not for binary masks).

translate/translation Translation correction (Ohser, 1983). Implemented for all window geometries, but slow for complex windows.

rigid Rigid motion correction (Ohser and Stoyan, 1981). Implemented for all window geometries, but slow for complex windows.

none Uncorrected estimate. An estimate of the K function *without* edge correction. (i.e. setting $e_{ij} = 1$ in the equation below. This estimate is **biased** and should not be used for data analysis, *unless* you have an extremely large point pattern (more than 100,000 points).

best Selects the best edge correction that is available for the geometry of the window. Currently this is Ripley's isotropic correction for a rectangular or polygonal window, and the translation correction for masks.

good Selects the best edge correction that can be computed in a reasonable time. This is the same as `"best"` for datasets with fewer than 3000 points; otherwise the selected edge correction is `"border"`, unless there are more than 100,000 points, when it is `"none"`.

The estimates of $K(r)$ are of the form

$$\hat{K}(r) = \frac{a}{n(n-1)} \sum_i \sum_j I(d_{ij} \le r) e_{ij}$$

where a is the area of the window, n is the number of data points, and the sum is taken over all ordered pairs of points i and j in `X`. Here d_{ij} is the distance between the two points, and $I(d_{ij} \le r)$

is the indicator that equals 1 if the distance is less than or equal to r. The term e_{ij} is the edge correction weight (which depends on the choice of edge correction listed above).

Note that this estimator assumes the process is stationary (spatially homogeneous). For inhomogeneous point patterns, see Kinhom.

If the point pattern X contains more than about 3000 points, the isotropic and translation edge corrections can be computationally prohibitive. The computations for the border method are much faster, and are statistically efficient when there are large numbers of points. Accordingly, if the number of points in X exceeds the threshold nlarge, then only the border correction will be computed. Setting nlarge=Inf or correction="best" will prevent this from happening. Setting nlarge=0 is equivalent to selecting only the border correction with correction="border".

If X contains more than about 100,000 points, even the border correction is time-consuming. You may want to consider setting correction="none" in this case. There is an even faster algorithm for the uncorrected estimate.

Approximations to the variance of $\hat{K}(r)$ are available, for the case of the isotropic edge correction estimator, **assuming complete spatial randomness** (Ripley, 1988; Lotwick and Silverman, 1982; Diggle, 2003, pp 51-53). If var.approx=TRUE, then the result of Kest also has a column named rip giving values of Ripley's (1988) approximation to var($\hat{K}(r)$), and (if the window is a rectangle) a column named ls giving values of Lotwick and Silverman's (1982) approximation.

If the argument domain is given, the calculations will be restricted to a subset of the data. In the formula for $K(r)$ above, the *first* point i will be restricted to lie inside domain. The result is an approximately unbiased estimate of $K(r)$ based on pairs of points in which the first point lies inside domain and the second point is unrestricted. This is useful in bootstrap techniques. The argument domain should be a window (object of class "owin") or something acceptable to as.owin. It must be a subset of the window of the point pattern X.

The estimator Kest ignores marks. Its counterparts for multitype point patterns are Kcross, Kdot, and for general marked point patterns see Kmulti.

Some writers, particularly Stoyan (1994, 1995) advocate the use of the "pair correlation function"

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$. See pcf on how to estimate this function.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Essentially a data frame containing columns

r	the vector of values of the argument r at which the function K has been estimated
theo	the theoretical value $K(r) = \pi r^2$ for a stationary Poisson process

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K(r)$ obtained by the edge corrections named.

If var.approx=TRUE then the return value also has columns rip and ls containing approximations to the variance of $\hat{K}(r)$ under CSR.

If ratio=TRUE then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $K(r)$.

Envelopes, significance bands and confidence intervals

To compute simulation envelopes for the K-function under CSR, use envelope.

To compute a confidence interval for the true K-function, use varblock or lohboot.

Warnings

The estimator of $K(r)$ is approximately unbiased for each fixed r. Bias increases with r and depends on the window geometry. For a rectangular window it is prudent to restrict the r values to a maximum of $1/4$ of the smaller side length of the rectangle. Bias may become appreciable for point patterns consisting of fewer than 15 points.

While $K(r)$ is always a non-decreasing function, the estimator of K is not guaranteed to be non-decreasing. This is rarely a problem in practice.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37–78.

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.

Ohser, J. and Stoyan, D. (1981) On the second-order and orientation analysis of planar stationary point processes. *Biometrical Journal* **23**, 523–533.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

localK to extract individual summands in the K function.

pcf for the pair correlation.

Fest, Gest, Jest for alternative summary functions.

Kcross, Kdot, Kinhom, Kmulti for counterparts of the K function for multitype point patterns.

reduced.sample for the calculation of reduced sample estimators.

Examples

```
X <- runifpoint(50)
K <- Kest(X)
K <- Kest(cells, correction="isotropic")
plot(K)
plot(K, main="K function for cells")
# plot the L function
plot(K, sqrt(iso/pi) ~ r)
plot(K, sqrt(./pi) ~ r, ylab="L(r)", main="L function for cells")
```

Kest.fft — *K-function using FFT*

Description

Estimates the reduced second moment function $K(r)$ from a point pattern in a window of arbitrary shape, using the Fast Fourier Transform.

Usage

```
Kest.fft(X, sigma, r=NULL, ..., breaks=NULL)
```

Arguments

`X`	The observed point pattern, from which an estimate of $K(r)$ will be computed. An object of class "ppp", or data in any format acceptable to `as.ppp()`.
`sigma`	Standard deviation of the isotropic Gaussian smoothing kernel.
`r`	Optional. Vector of values for the argument r at which $K(r)$ should be evaluated. There is a sensible default.
`...`	Arguments passed to `as.mask` determining the spatial resolution for the FFT calculation.
`breaks`	This argument is for internal use only.

Details

This is an alternative to the function `Kest` for estimating the K function. It may be useful for very large patterns of points.

Whereas `Kest` computes the distance between each pair of points analytically, this function discretises the point pattern onto a rectangular pixel raster and applies Fast Fourier Transform techniques to estimate $K(t)$. The hard work is done by the function `Kmeasure`.

The result is an approximation whose accuracy depends on the resolution of the pixel raster. The resolution is controlled by the arguments `...`, or by setting the parameter `npixel` in `spatstat.options`.

Value

An object of class "fv" (see `fv.object`).

Essentially a data frame containing columns

`r`	the vector of values of the argument r at which the function K has been estimated
`border`	the estimates of $K(r)$ for these values of r
`theo`	the theoretical value $K(r) = \pi r^2$ for a stationary Poisson process

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

Kest, Kmeasure, spatstat.options

Examples

```
pp <- runifpoint(10000)

Kpp <- Kest.fft(pp, 0.01)
plot(Kpp)
```

Kinhom *Inhomogeneous K-function*

Description

Estimates the inhomogeneous K function of a non-stationary point pattern.

Usage

```
Kinhom(X, lambda=NULL, ..., r = NULL, breaks = NULL,
  correction=c("border", "bord.modif", "isotropic", "translate"),
  renormalise=TRUE,
  normpower=1,
  update=TRUE,
  leaveoneout=TRUE,
  nlarge = 1000,
  lambda2=NULL, reciplambda=NULL, reciplambda2=NULL,
  diagonal=TRUE,
  sigma=NULL, varcov=NULL,
  ratio=FALSE)
```

Arguments

`X`	The observed data point pattern, from which an estimate of the inhomogeneous K function will be computed. An object of class `"ppp"` or in a format recognised by `as.ppp()`
`lambda`	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern `X`, a pixel image (object of class `"im"`) giving the intensity values at all locations, a fitted point process model (object of class `"ppm"` or `"kppm"`) or a `function(x,y)` which can be evaluated to give the intensity value at any location.
`...`	Extra arguments. Ignored if `lambda` is present. Passed to `density.ppp` if `lambda` is omitted.
`r`	vector of values for the argument r at which the inhomogeneous K function should be evaluated. Not normally given by the user; there is a sensible default.
`breaks`	This argument is for internal use only.
`correction`	A character vector containing any selection of the options `"border"`, `"bord.modif"`, `"isotropic"`, `"Ripley"`, `"translate"`, `"translation"`, `"none"` or `"best"`. It specifies the edge correction(s) to be applied. Alternatively `correction="all"` selects all options.
`renormalise`	Logical. Whether to renormalise the estimate. See Details.
`normpower`	Integer (usually either 1 or 2). Normalisation power. See Details.
`update`	Logical value indicating what to do when `lambda` is a fitted model (class `"ppm"`, `"kppm"` or `"dppm"`). If `update=TRUE` (the default), the model will first be refitted to the data `X` (using `update.ppm` or `update.kppm`) before the fitted intensity is computed. If `update=FALSE`, the fitted intensity of the model will be computed without re-fitting it to `X`.
`leaveoneout`	Logical value (passed to `density.ppp` or `fitted.ppm`) specifying whether to use a leave-one-out rule when calculating the intensity.
`nlarge`	Optional. Efficiency threshold. If the number of points exceeds `nlarge`, then only the border correction will be computed, using a fast algorithm.
`lambda2`	Advanced use only. Matrix containing estimates of the products $\lambda(x_i)\lambda(x_j)$ of the intensities at each pair of data points x_i and x_j.
`reciplambda`	Alternative to `lambda`. Values of the estimated *reciprocal* $1/\lambda$ of the intensity function. Either a vector giving the reciprocal intensity values at the points of the pattern `X`, a pixel image (object of class `"im"`) giving the reciprocal intensity values at all locations, or a `function(x,y)` which can be evaluated to give the reciprocal intensity value at any location.
`reciplambda2`	Advanced use only. Alternative to `lambda2`. A matrix giving values of the estimated *reciprocal products* $1/\lambda(x_i)\lambda(x_j)$ of the intensities at each pair of data points x_i and x_j.
`diagonal`	Do not use this argument.
`sigma,varcov`	Optional arguments passed to `density.ppp` to control the smoothing bandwidth, when `lambda` is estimated by kernel smoothing.
`ratio`	Logical. If `TRUE`, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

This computes a generalisation of the K function for inhomogeneous point patterns, proposed by Baddeley, Møller and Waagepetersen (2000).

The "ordinary" K function (variously known as the reduced second order moment function and Ripley's K function), is described under `Kest`. It is defined only for stationary point processes.

The inhomogeneous K function $K_{\text{inhom}}(r)$ is a direct generalisation to nonstationary point processes. Suppose x is a point process with non-constant intensity $\lambda(u)$ at each location u. Define $K_{\text{inhom}}(r)$ to be the expected value, given that u is a point of x, of the sum of all terms $1/\lambda(x_j)$ over all points x_j in the process separated from u by a distance less than r. This reduces to the ordinary K function if $\lambda()$ is constant. If x is an inhomogeneous Poisson process with intensity function $\lambda(u)$, then $K_{\text{inhom}}(r) = \pi r^2$.

Given a point pattern dataset, the inhomogeneous K function can be estimated essentially by summing the values $1/(\lambda(x_i)\lambda(x_j))$ for all pairs of points x_i, x_j separated by a distance less than r.

This allows us to inspect a point pattern for evidence of interpoint interactions after allowing for spatial inhomogeneity of the pattern. Values $K_{\text{inhom}}(r) > \pi r^2$ are suggestive of clustering.

The argument `lambda` should supply the (estimated) values of the intensity function λ. It may be either

a numeric vector containing the values of the intensity function at the points of the pattern `X`.

a pixel image (object of class `"im"`) assumed to contain the values of the intensity function at all locations in the window.

a fitted point process model (object of class `"ppm"`, `"kppm"` or `"dppm"`) whose fitted *trend* can be used as the fitted intensity. (If `update=TRUE` the model will first be refitted to the data `X` before the trend is computed.)

a function which can be evaluated to give values of the intensity at any locations.

omitted: if `lambda` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

If `lambda` is a numeric vector, then its length should be equal to the number of points in the pattern `X`. The value `lambda[i]` is assumed to be the the (estimated) value of the intensity $\lambda(x_i)$ for the point x_i of the pattern X. Each value must be a positive number; `NA`'s are not allowed.

If `lambda` is a pixel image, the domain of the image should cover the entire window of the point pattern. If it does not (which may occur near the boundary because of discretisation error), then the missing pixel values will be obtained by applying a Gaussian blur to `lambda` using `blur`, then looking up the values of this blurred image for the missing locations. (A warning will be issued in this case.)

If `lambda` is a function, then it will be evaluated in the form `lambda(x,y)` where `x` and `y` are vectors of coordinates of the points of `X`. It should return a numeric vector with length equal to the number of points in `X`.

If `lambda` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate `lambda[i]` for the point `X[i]` is computed by removing `X[i]` from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point `X[i]`. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Edge corrections are used to correct bias in the estimation of K_{inhom}. Each edge-corrected estimate of $K_{\text{inhom}}(r)$ is of the form

$$\widehat{K}_{\text{inhom}}(r) = (1/A) \sum_i \sum_j \frac{1\{d_{ij} \le r\} e(x_i, x_j, r)}{\lambda(x_i)\lambda(x_j)}$$

where A is a constant denominator, d_{ij} is the distance between points x_i and x_j, and $e(x_i, x_j, r)$ is an edge correction factor. For the 'border' correction,

$$e(x_i, x_j, r) = \frac{1(b_i > r)}{\sum_j 1(b_j > r)/\lambda(x_j)}$$

where b_i is the distance from x_i to the boundary of the window. For the 'modified border' correction,

$$e(x_i, x_j, r) = \frac{1(b_i > r)}{\text{area}(W \ominus r)}$$

where $W \ominus r$ is the eroded window obtained by trimming a margin of width r from the border of the original window. For the 'translation' correction,

$$e(x_i, x_j, r) = \frac{1}{\text{area}(W \cap (W + (x_j - x_i)))}$$

and for the 'isotropic' correction,

$$e(x_i, x_j, r) = \frac{1}{\text{area}(W) g(x_i, x_j)}$$

where $g(x_i, x_j)$ is the fraction of the circumference of the circle with centre x_i and radius $||x_i - x_j||$ which lies inside the window.

If `renormalise=TRUE` (the default), then the estimates described above are multiplied by $c^{\text{normpower}}$ where $c = \text{area}(W)/\sum(1/\lambda(x_i))$. This rescaling reduces the variability and bias of the estimate in small samples and in cases of very strong inhomogeneity. The default value of `normpower` is 1 (for consistency with previous versions of **spatstat**) but the most sensible value is 2, which would correspond to rescaling the `lambda` values so that $\sum(1/\lambda(x_i)) = \text{area}(W)$.

If the point pattern `X` contains more than about 1000 points, the isotropic and translation edge corrections can be computationally prohibitive. The computations for the border method are much faster, and are statistically efficient when there are large numbers of points. Accordingly, if the number of points in `X` exceeds the threshold `nlarge`, then only the border correction will be computed. Setting `nlarge=Inf` or `correction="best"` will prevent this from happening. Setting `nlarge=0` is equivalent to selecting only the border correction with `correction="border"`.

The pair correlation function can also be applied to the result of `Kinhom`; see `pcf`.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing at least the following columns,

`r`	the vector of values of the argument r at which $K_{\text{inhom}}(r)$ has been estimated
`theo`	vector of values of πr^2, the theoretical value of $K_{\text{inhom}}(r)$ for an inhomogeneous Poisson process

and containing additional columns according to the choice specified in the `correction` argument. The additional columns are named `border`, `trans` and `iso` and give the estimated values of $K_{\text{inhom}}(r)$ using the border correction, translation correction, and Ripley isotropic correction, respectively.

If `ratio=TRUE` then the return value also has two attributes called `"numerator"` and `"denominator"` which are `"fv"` objects containing the numerators and denominators of each estimate of $K_{\text{inhom}}(r)$.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Møller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

See Also

Kest, pcf

Examples

```
  # inhomogeneous pattern of maples
  X <- unmark(split(lansing)$maple)


  # (1) intensity function estimated by model-fitting
  # Fit spatial trend: polynomial in x and y coordinates
  fit <- ppm(X, ~ polynom(x,y,2), Poisson())
  # (a) predict intensity values at points themselves,
  #     obtaining a vector of lambda values
  lambda <- predict(fit, locations=X, type="trend")
  # inhomogeneous K function
  Ki <- Kinhom(X, lambda)
  plot(Ki)
  # (b) predict intensity at all locations,
  #     obtaining a pixel image
  lambda <- predict(fit, type="trend")
  Ki <- Kinhom(X, lambda)
  plot(Ki)

  # (2) intensity function estimated by heavy smoothing
  Ki <- Kinhom(X, sigma=0.1)
  plot(Ki)

  # (3) simulated data: known intensity function
  lamfun <- function(x,y) { 50 + 100 * x }
  # inhomogeneous Poisson process
  Y <- rpoispp(lamfun, 150, owin())
  # inhomogeneous K function
  Ki <- Kinhom(Y, lamfun)
  plot(Ki)

  # How to make simulation envelopes:
  #      Example shows method (2)
  ## Not run:
  smo <- density.ppp(X, sigma=0.1)
  Ken <- envelope(X, Kinhom, nsim=99,
                  simulate=expression(rpoispp(smo)),
                  sigma=0.1, correction="trans")
  plot(Ken)

## End(Not run)
```

km.rs *Kaplan-Meier and Reduced Sample Estimator using Histograms*

Description

Compute the Kaplan-Meier and Reduced Sample estimators of a survival time distribution function, using histogram techniques

Usage

```
km.rs(o, cc, d, breaks)
```

Arguments

`o`	vector of observed survival times
`cc`	vector of censoring times
`d`	vector of non-censoring indicators
`breaks`	Vector of breakpoints to be used to form histograms.

Details

This function is needed mainly for internal use in **spatstat**, but may be useful in other applications where you want to form the Kaplan-Meier estimator from a huge dataset.

Suppose T_i are the survival times of individuals $i = 1, \ldots, M$ with unknown distribution function $F(t)$ which we wish to estimate. Suppose these times are right-censored by random censoring times C_i. Thus the observations consist of right-censored survival times $\tilde{T}_i = \min(T_i, C_i)$ and non-censoring indicators $D_i = 1\{T_i \le C_i\}$ for each i.

The arguments to this function are vectors `o`, `cc`, `d` of observed values of $\tilde{T}_i$, C_i and D_i respectively. The function computes histograms and forms the reduced-sample and Kaplan-Meier estimates of $F(t)$ by invoking the functions `kaplan.meier` and `reduced.sample`. This is efficient if the lengths of `o`, `cc`, `d` (i.e. the number of observations) is large.

The vectors `km` and `hazard` returned by `kaplan.meier` are (histogram approximations to) the Kaplan-Meier estimator of $F(t)$ and its hazard rate $\lambda(t)$. Specifically, `km[k]` is an estimate of `F(breaks[k+1])`, and `lambda[k]` is an estimate of the average of $\lambda(t)$ over the interval `(breaks[k],breaks[k+1])`. This approximation is exact only if the survival times are discrete and the histogram breaks are fine enough to ensure that each interval `(breaks[k],breaks[k+1])` contains only one possible value of the survival time.

The vector `rs` is the reduced-sample estimator, `rs[k]` being the reduced sample estimate of `F(breaks[k+1])`. This value is exact, i.e. the use of histograms does not introduce any approximation error in the reduced-sample estimator.

Value

A list with five elements

`rs`	Reduced-sample estimate of the survival time c.d.f. $F(t)$
`km`	Kaplan-Meier estimate of the survival time c.d.f. $F(t)$
`hazard`	corresponding Nelson-Aalen estimate of the hazard rate $\lambda(t)$
`r`	values of t for which $F(t)$ is estimated
`breaks`	the breakpoints vector

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

reduced.sample, kaplan.meier

Kmark *Mark-Weighted K Function*

Description

Estimates the mark-weighted K function of a marked point pattern.

Usage

```
Kmark(X, f = NULL, r = NULL,
      correction = c("isotropic", "Ripley", "translate"), ...,
      f1 = NULL, normalise = TRUE, returnL = FALSE, fargs = NULL)

markcorrint(X, f = NULL, r = NULL,
            correction = c("isotropic", "Ripley", "translate"), ...,
            f1 = NULL, normalise = TRUE, returnL = FALSE, fargs = NULL)
```

Arguments

X	The observed point pattern. An object of class "ppp" or something acceptable to as.ppp.
f	Optional. Test function f used in the definition of the mark correlation function. An R function with at least two arguments. There is a sensible default.
r	Optional. Numeric vector. The values of the argument r at which the mark correlation function $k_f(r)$ should be evaluated. There is a sensible default.
correction	A character vector containing any selection of the options "isotropic", "Ripley" or "translate". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
...	Ignored.
f1	An alternative to f. If this argument is given, then f is assumed to take the form $f(u,v) = f_1(u)f_1(v)$.
normalise	If normalise=FALSE, compute only the numerator of the expression for the mark correlation.
returnL	Compute the analogue of the K-function if returnL=FALSE or the analogue of the L-function if returnL=TRUE.
fargs	Optional. A list of extra arguments to be passed to the function f or f1.

Details

The functions `Kmark` and `markcorrint` are identical. (Eventually `markcorrint` will be deprecated.)

The *mark-weighted K function* $K_f(r)$ of a marked point process (Penttinen et al, 1992) is a generalisation of Ripley's K function, in which the contribution from each pair of points is weighted by a function of their marks. If the marks of the two points are m_1, m_2 then the weight is proportional to $f(m_1, m_2)$ where f is a specified *test function*.

The mark-weighted K function is defined so that

$$\lambda K_f(r) = \frac{C_f(r)}{E[f(M_1, M_2)]}$$

where

$$C_f(r) = E\left[\sum_{x \in X} f(m(u), m(x)) 10 < ||u - x|| \leq r \;\Big|\; u \in X\right]$$

for any spatial location u taken to be a typical point of the point process X. Here $||u - x||$ is the euclidean distance between u and x, so that the sum is taken over all random points x that lie within a distance r of the point u. The function $C_f(r)$ is the *unnormalised* mark-weighted K function. To obtain $K_f(r)$ we standardise $C_f(r)$ by dividing by $E[f(M_1, M_2)]$, the expected value of $f(M_1, M_2)$ when M_1 and M_2 are independent random marks with the same distribution as the marks in the point process.

Under the hypothesis of random labelling, the mark-weighted K function is equal to Ripley's K function, $K_f(r) = K(r)$.

The mark-weighted K function is sometimes called the *mark correlation integral* because it is related to the mark correlation function $k_f(r)$ and the pair correlation function $g(r)$ by

$$K_f(r) = 2\pi \int_0^r s k_f(s)\, g(s)\, \mathrm{d}s$$

See `markcorr` for a definition of the mark correlation function.

Given a marked point pattern `X`, this command computes edge-corrected estimates of the mark-weighted K function. If `returnL=FALSE` then the estimated function $K_f(r)$ is returned; otherwise the function

$$L_f(r) = \sqrt{K_f(r)/\pi}$$

is returned.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing numeric columns

`r`	the values of the argument r at which the mark correlation integral $K_f(r)$ has been estimated
`theo`	the theoretical value of $K_f(r)$ when the marks attached to different points are independent, namely πr^2

together with a column or columns named `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the mark-weighted K function $K_f(r)$ obtained by the edge corrections named (if `returnL=FALSE`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Penttinen, A., Stoyan, D. and Henttonen, H. M. (1992) Marked point processes in forest statistics. *Forest Science* **38** (1992) 806-824.

Illian, J., Penttinen, A., Stoyan, H. and Stoyan, D. (2008) *Statistical analysis and modelling of spatial point patterns*. Chichester: John Wiley.

See Also

markcorr to estimate the mark correlation function.

Examples

```
# CONTINUOUS-VALUED MARKS:
# (1) Spruces
# marks represent tree diameter
# mark correlation function
ms <- Kmark(spruces)
plot(ms)

# (2) simulated data with independent marks
X <- rpoispp(100)
X <- X %mark% runif(npoints(X))
Xc <- Kmark(X)
plot(Xc)

# MULTITYPE DATA:
# Hughes' amacrine data
# Cells marked as 'on'/'off'
M <- Kmark(amacrine, function(m1,m2) {m1==m2},
                     correction="translate")
plot(M)
```

Kmeasure *Reduced Second Moment Measure*

Description

Estimates the reduced second moment measure κ from a point pattern in a window of arbitrary shape.

Usage

```
Kmeasure(X, sigma, edge=TRUE, ..., varcov=NULL)
```

Arguments

`X`	The observed point pattern, from which an estimate of κ will be computed. An object of class `"ppp"`, or data in any format acceptable to `as.ppp()`.
`sigma`	Standard deviation σ of the Gaussian smoothing kernel. Incompatible with `varcov`.
`edge`	Logical value indicating whether an edge correction should be applied.
`...`	Arguments passed to `as.mask` controlling the pixel resolution.
`varcov`	Variance-covariance matrix of the Gaussian smoothing kernel. Incompatible with `sigma`.

Details

Given a point pattern dataset, this command computes an estimate of the reduced second moment measure κ of the point process. The result is a pixel image whose pixel values are estimates of the density of the reduced second moment measure.

The reduced second moment measure κ can be regarded as a generalisation of the more familiar K-function. An estimate of κ derived from a spatial point pattern dataset can be useful in exploratory data analysis. Its advantage over the K-function is that it is also sensitive to anisotropy and directional effects.

In a nutshell, the command `Kmeasure` computes a smoothed version of the *Fry plot*. As explained under `fryplot`, the Fry plot is a scatterplot of the vectors joining all pairs of points in the pattern. The reduced second moment measure is (essentially) defined as the average of the Fry plot over different realisations of the point process. The command `Kmeasure` effectively smooths the Fry plot of a dataset to obtain an estimate of the reduced second moment measure.

In formal terms, the reduced second moment measure κ of a stationary point process X is a measure defined on the two-dimensional plane such that, for a 'typical' point x of the process, the expected number of other points y of the process such that the vector $y - x$ lies in a region A, equals $\lambda\kappa(A)$. Here λ is the intensity of the process, i.e. the expected number of points of X per unit area.

The K-function is a special case. The function value $K(t)$ is the value of the reduced second moment measure for the disc of radius t centred at the origin; that is, $K(t) = \kappa(b(0, t))$.

The command `Kmeasure` computes an estimate of κ from a point pattern dataset `X`, which is assumed to be a realisation of a stationary point process, observed inside a known, bounded window. Marks are ignored.

The algorithm approximates the point pattern and its window by binary pixel images, introduces a Gaussian smoothing kernel and uses the Fast Fourier Transform `fft` to form a density estimate of κ. The calculation corresponds to the edge correction known as the "translation correction".

The Gaussian smoothing kernel may be specified by either of the arguments `sigma` or `varcov`. If `sigma` is a single number, this specifies an isotropic Gaussian kernel with standard deviation `sigma` on each coordinate axis. If `sigma` is a vector of two numbers, this specifies a Gaussian kernel with standard deviation `sigma[1]` on the x axis, standard deviation `sigma[2]` on the y axis, and zero correlation between the x and y axes. If `varcov` is given, this specifies the variance-covariance matrix of the Gaussian kernel. There do not seem to be any well-established rules for selecting the smoothing kernel in this context.

The density estimate of κ is returned in the form of a real-valued pixel image. Pixel values are estimates of the normalised second moment density at the centre of the pixel. (The uniform Poisson process would have values identically equal to 1.) The image `x` and `y` coordinates are on the same scale as vector displacements in the original point pattern window. The point `x=0, y=0` corresponds to the 'typical point'. A peak in the image near `(0,0)` suggests clustering; a dip in the image near `(0,0)` suggests inhibition; peaks or dips at other positions suggest possible periodicity.

If desired, the value of $\kappa(A)$ for a region A can be estimated by computing the integral of the pixel image over the domain A, i.e.\ summing the pixel values and multiplying by pixel area, using `integral.im`. One possible application is to compute anisotropic counterparts of the K-function (in which the disc of radius t is replaced by another shape). See Examples.

Value

A real-valued pixel image (an object of class `"im"`, see `im.object`) whose pixel values are estimates of the density of the reduced second moment measure at each location.

Warning

Some writers use the term *reduced second moment measure* when they mean the K-function. This has caused confusion.

As originally defined, the reduced second moment measure is a measure, obtained by modifying the second moment measure, while the K-function is a function obtained by evaluating this measure for discs of increasing radius. In **spatstat**, the K-function is computed by `Kest` and the reduced second moment measure is computed by `Kmeasure`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

`Kest`, `fryplot`, `spatstat.options`, `integral.im`, `im.object`

Examples

```
data(cells)
plot(Kmeasure(cells, 0.05))
# shows pronounced dip around origin consistent with strong inhibition
data(redwood)
plot(Kmeasure(redwood, 0.03), col=grey(seq(1,0,length=32)))
# shows peaks at several places, reflecting clustering and ?periodicity
M <- Kmeasure(cells, 0.05)
# evaluate measure on a sector
W <- Window(M)
ang <- as.im(atan2, W)
rad <- as.im(function(x,y){sqrt(x^2+y^2)}, W)
sector <- solutionset(ang > 0 & ang < 1 & rad < 0.6)
integral.im(M[sector, drop=FALSE])
```

`Kmodel` *K Function or Pair Correlation Function of a Point Process Model*

Description

Returns the theoretical K function or the pair correlation function of a point process model.

Usage

```
Kmodel(model, ...)

pcfmodel(model, ...)
```

Arguments

`model`	A fitted point process model of some kind.
`...`	Ignored.

Details

For certain types of point process models, it is possible to write down a mathematical expression for the K function or the pair correlation function of the model.

The functions `Kmodel` and `pcfmodel` give the theoretical K-function and the theoretical pair correlation function for a point process model that has been fitted to data.

The functions `Kmodel` and `pcfmodel` are generic, with methods for the classes `"kppm"` (cluster processes and Cox processes) and `"ppm"` (Gibbs processes).

The return value is a `function` in the R language, which takes one argument `r`. Evaluation of this function, on a numeric vector `r`, yields values of the desired K function or pair correlation function at these distance values.

Value

A `function` in the R language, which takes one argument `r`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`Kest` or `pcf` to estimate the K function or pair correlation function nonparametrically from data.

`Kmodel.kppm` for the method for cluster processes and Cox processes.

`Kmodel.ppm` for the method for Gibbs processes.

Kmodel.dppm *K-function or Pair Correlation Function of a Determinantal Point Process Model*

Description

Returns the theoretical K-function or theoretical pair correlation function of a determinantal point process model as a function of one argument r.

Usage

```
   ## S3 method for class 'dppm'
Kmodel(model, ...)

   ## S3 method for class 'dppm'
pcfmodel(model, ...)

   ## S3 method for class 'detpointprocfamily'
Kmodel(model, ...)

   ## S3 method for class 'detpointprocfamily'
pcfmodel(model, ...)
```

Arguments

`model`	Model of class "detpointprocfamily" or "dppm".
`...`	Ignored (not quite true – there is some undocumented internal use)

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
model <- dppMatern(lambda=100, alpha=.01, nu=1, d=2)
KMatern <- Kmodel(model)
pcfMatern <- pcfmodel(model)
plot(KMatern, xlim = c(0,0.05))
plot(pcfMatern, xlim = c(0,0.05))
```

Kmodel.kppm — *K Function or Pair Correlation Function of Cluster Model or Cox model*

Description

Returns the theoretical K function or the pair correlation function of a cluster point process model or Cox point process model.

Usage

```
## S3 method for class 'kppm'
Kmodel(model, ...)

## S3 method for class 'kppm'
pcfmodel(model, ...)
```

Arguments

`model`	A fitted cluster point process model (object of class `"kppm"`) typically obtained from the model-fitting algorithm `kppm`.
`...`	Ignored.

Details

For certain types of point process models, it is possible to write down a mathematical expression for the K function or the pair correlation function of the model. In particular this is possible for a fitted cluster point process model (object of class `"kppm"` obtained from `kppm`).

The functions `Kmodel` and `pcfmodel` are generic. The functions documented here are the methods for the class `"kppm"`.

The return value is a `function` in the R language, which takes one argument `r`. Evaluation of this function, on a numeric vector `r`, yields values of the desired K function or pair correlation function at these distance values.

Value

A `function` in the R language, which takes one argument `r`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`Kest` or `pcf` to estimate the K function or pair correlation function nonparametrically from data.

`kppm` to fit cluster models.

`Kmodel` for the generic functions.

`Kmodel.ppm` for the method for Gibbs processes.

Examples

```
data(redwood)
fit <- kppm(redwood, ~x, "MatClust")
K <- Kmodel(fit)
K(c(0.1, 0.2))
curve(K(x), from=0, to=0.25)
```

Kmodel.ppm *K Function or Pair Correlation Function of Gibbs Point Process model*

Description

Returns the theoretical K function or the pair correlation function of a fitted Gibbs point process model.

Usage

```
## S3 method for class 'ppm'
Kmodel(model, ...)

## S3 method for class 'ppm'
pcfmodel(model, ...)
```

Arguments

model	A fitted Poisson or Gibbs point process model (object of class "ppm") typically obtained from the model-fitting algorithm ppm.
...	Ignored.

Details

This function computes an *approximation* to the K function or the pair correlation function of a Gibbs point process.

The functions Kmodel and pcfmodel are generic. The functions documented here are the methods for the class "ppm".

The approximation is only available for stationary pairwise-interaction models. It uses the second order Poisson-saddlepoint approximation (Baddeley and Nair, 2012b) which is a combination of the Poisson-Boltzmann-Emden and Percus-Yevick approximations.

The return value is a function in the R language, which takes one argument r. Evaluation of this function, on a numeric vector r, yields values of the desired K function or pair correlation function at these distance values.

Value

A function in the R language, which takes one argument r.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Gopalan Nair.

References

Baddeley, A. and Nair, G. (2012a) Fast approximation of the intensity of Gibbs point processes. *Electronic Journal of Statistics* **6** 1155–1169.

Baddeley, A. and Nair, G. (2012b) Approximating the moments of a spatial point process. *Stat* **1**, 1, 18–30. doi: 10.1002/sta4.5

See Also

`Kest` or `pcf` to estimate the K function or pair correlation function nonparametrically from data.

`ppm` to fit Gibbs models.

`Kmodel` for the generic functions.

`Kmodel.kppm` for the method for cluster/Cox processes.

Examples

```
fit <- ppm(swedishpines, ~1, Strauss(8))
p <- pcfmodel(fit)
K <- Kmodel(fit)
p(6)
K(8)
curve(K(x), from=0, to=15)
```

`Kmulti` *Marked K-Function*

Description

For a marked point pattern, estimate the multitype K function which counts the expected number of points of subset J within a given distance from a typical point in subset `I`.

Usage

```
Kmulti(X, I, J, r=NULL, breaks=NULL, correction, ..., ratio=FALSE)
```

Arguments

`X`	The observed point pattern, from which an estimate of the multitype K function $K_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
`I`	Subset index specifying the points of `X` from which distances are measured. See Details.
`J`	Subset index specifying the points in `X` to which distances are measured. See Details.
`r`	numeric vector. The values of the argument r at which the multitype K function $K_{IJ}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`breaks`	This argument is for internal use only.

correction
: A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.

...
: Ignored.

ratio
: Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

The function Kmulti generalises Kest (for unmarked point patterns) and Kdot and Kcross (for multitype point patterns) to arbitrary marked point patterns.

Suppose X_I, X_J are subsets, possibly overlapping, of a marked point process. The multitype K function is defined so that $\lambda_J K_{IJ}(r)$ equals the expected number of additional random points of X_J within a distance r of a typical point of X_I. Here λ_J is the intensity of X_J i.e. the expected number of points of X_J per unit area. The function K_{IJ} is determined by the second order moment properties of X.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp.

The arguments I and J specify two subsets of the point pattern. They may be any type of subset indices, for example, logical vectors of length equal to npoints(X), or integer vectors with entries in the range 1 to npoints(X), or negative integer vectors.

Alternatively, I and J may be **functions** that will be applied to the point pattern X to obtain index vectors. If I is a function, then evaluating I(X) should yield a valid subset index. This option is useful when generating simulation envelopes using envelope.

The argument r is the vector of values for the distance r at which $K_{IJ}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of hist) for the computation of histograms of distances.

First-time users would be strongly advised not to specify r. However, if it is specified, r must satisfy r[1] = 0, and max(r) must be larger than the radius of the largest disc contained in the window.

This algorithm assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape.

Biases due to edge effects are treated in the same manner as in Kest. The edge corrections implemented here are

border the border method or "reduced sample" estimator (see Ripley, 1988). This is the least efficient (statistically) and the fastest to compute. It can be computed for a window of arbitrary shape.

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is currently implemented only for rectangular and polygonal windows.

translate Translation correction (Ohser, 1983). Implemented for all window geometries.

The pair correlation function pcf can also be applied to the result of Kmulti.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $K_{IJ}(r)$ has been estimated
theo	the theoretical value of $K_{IJ}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{IJ}(r)$ obtained by the edge corrections named.

If ratio=TRUE then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $K(r)$.

Warnings

The function K_{IJ} is not necessarily differentiable.

The border correction (reduced sample) estimator of K_{IJ} used here is pointwise approximately unbiased, but need not be a nondecreasing function of r, while the true K_{IJ} must be nondecreasing.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.

Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

Kcross, Kdot, Kest, pcf

Examples

```
 # Longleaf Pine data: marks represent diameter
trees <- longleaf

K <- Kmulti(trees, marks(trees) <= 15, marks(trees) >= 25)
plot(K)
# functions determining subsets
f1 <- function(X) { marks(X) <= 15 }
f2 <- function(X) { marks(X) >= 15 }
K <- Kmulti(trees, f1, f2)
```

Kmulti.inhom *Inhomogeneous Marked K-Function*

Description

For a marked point pattern, estimate the inhomogeneous version of the multitype K function which counts the expected number of points of subset J within a given distance from a typical point in subset I, adjusted for spatially varying intensity.

Usage

```
Kmulti.inhom(X, I, J, lambdaI=NULL, lambdaJ=NULL,
         ...,
         r=NULL, breaks=NULL,
         correction=c("border", "isotropic", "Ripley", "translate"),
         lambdaIJ=NULL,
         sigma=NULL, varcov=NULL,
         lambdaX=NULL, update=TRUE, leaveoneout=TRUE)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous multitype K function $K_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
I	Subset index specifying the points of X from which distances are measured. See Details.
J	Subset index specifying the points in X to which distances are measured. See Details.
lambdaI	Optional. Values of the estimated intensity of the sub-process X[I]. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X[I], a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location,
lambdaJ	Optional. Values of the estimated intensity of the sub-process X[J]. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X[J], a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
...	Ignored.
r	Optional. Numeric vector. The values of the argument r at which the multitype K function $K_{IJ}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.

lambdaIJ: Optional. A matrix containing estimates of the product of the intensities `lambdaI` and `lambdaJ` for each pair of points, the first point belonging to subset `I` and the second point to subset `J`.

sigma,varcov: Optional arguments passed to `density.ppp` to control the smoothing bandwidth, when `lambda` is estimated by kernel smoothing.

lambdaX: Optional. Values of the intensity for all points of `X`. Either a pixel image (object of class `"im"`), a numeric vector containing the intensity values at each of the points in `X`, a fitted point process model (object of class `"ppm"` or `"kppm"` or `"dppm"`), or a `function(x,y)` which can be evaluated to give the intensity value at any location. If present, this argument overrides both `lambdaI` and `lambdaJ`.

update: Logical value indicating what to do when `lambdaI`, `lambdaJ` or `lambdaX` is a fitted point process model (class `"ppm"`, `"kppm"` or `"dppm"`). If `update=TRUE` (the default), the model will first be refitted to the data `X` (using `update.ppm` or `update.kppm`) before the fitted intensity is computed. If `update=FALSE`, the fitted intensity of the model will be computed without re-fitting it to `X`.

leaveoneout: Logical value (passed to `density.ppp` or `fitted.ppm`) specifying whether to use a leave-one-out rule when calculating the intensity.

Details

The function `Kmulti.inhom` is the counterpart, for spatially-inhomogeneous marked point patterns, of the multitype K function `Kmulti`.

Suppose X is a marked point process, with marks of any kind. Suppose X_I, X_J are two sub-processes, possibly overlapping. Typically X_I would consist of those points of X whose marks lie in a specified range of mark values, and similarly for X_J. Suppose that $\lambda_I(u)$, $\lambda_J(u)$ are the spatially-varying intensity functions of X_I and X_J respectively. Consider all the pairs of points (u, v) in the point process X such that the first point u belongs to X_I, the second point v belongs to X_J, and the distance between u and v is less than a specified distance r. Give this pair (u, v) the numerical weight $1/(\lambda_I(u)\lambda_J(u))$. Calculate the sum of these weights over all pairs of points as described. This sum (after appropriate edge-correction and normalisation) is the estimated inhomogeneous multitype K function.

The argument `X` must be a point pattern (object of class `"ppp"`) or any data that are acceptable to `as.ppp`.

The arguments `I` and `J` specify two subsets of the point pattern. They may be any type of subset indices, for example, logical vectors of length equal to `npoints(X)`, or integer vectors with entries in the range 1 to `npoints(X)`, or negative integer vectors.

Alternatively, `I` and `J` may be **functions** that will be applied to the point pattern `X` to obtain index vectors. If `I` is a function, then evaluating `I(X)` should yield a valid subset index. This option is useful when generating simulation envelopes using `envelope`.

The argument `lambdaI` supplies the values of the intensity of the sub-process identified by index `I`. It may be either

a pixel image (object of class `"im"`) which gives the values of the intensity of `X[I]` at all locations in the window containing `X`;

a numeric vector containing the values of the intensity of `X[I]` evaluated only at the data points of `X[I]`. The length of this vector must equal the number of points in `X[I]`.

a function of the form `function(x,y)` which can be evaluated to give values of the intensity at any locations.

a fitted point process model (object of class "ppm", "kppm" or "dppm") whose fitted *trend* can be used as the fitted intensity. (If `update=TRUE` the model will first be refitted to the data `X` before the trend is computed.)

omitted: if `lambdaI` is omitted then it will be estimated using a leave-one-out kernel smoother.

If `lambdaI` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate of `lambdaI` for a given point is computed by removing the point from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point in question. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Similarly `lambdaJ` supplies the values of the intensity of the sub-process identified by index `J`.

Alternatively if the argument `lambdaX` is given, then it specifies the intensity values for all points of `X`, and the arguments `lambdaI`, `lambdaJ` will be ignored.

The argument `r` is the vector of values for the distance r at which $K_{IJ}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of `hist`) for the computation of histograms of distances.

First-time users would be strongly advised not to specify `r`. However, if it is specified, `r` must satisfy `r[1] = 0`, and `max(r)` must be larger than the radius of the largest disc contained in the window.

Biases due to edge effects are treated in the same manner as in `Kinhom`. The edge corrections implemented here are

border the border method or "reduced sample" estimator (see Ripley, 1988). This is the least efficient (statistically) and the fastest to compute. It can be computed for a window of arbitrary shape.

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is currently implemented only for rectangular windows.

translate Translation correction (Ohser, 1983). Implemented for all window geometries.

The pair correlation function `pcf` can also be applied to the result of `Kmulti.inhom`.

Value

An object of class "fv" (see `fv.object`).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $K_{IJ}(r)$ has been estimated
theo	the theoretical value of $K_{IJ}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{IJ}(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Møller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

See Also

Kmulti, Kdot.inhom, Kcross.inhom, pcf

Examples

```
# Finnish Pines data: marked by diameter and height
plot(finpines, which.marks="height")
II <- (marks(finpines)$height <= 2)
JJ <- (marks(finpines)$height > 3)
K <- Kmulti.inhom(finpines, II, JJ)
plot(K)
# functions determining subsets
f1 <- function(X) { marks(X)$height <= 2 }
f2 <- function(X) { marks(X)$height > 3 }
K <- Kmulti.inhom(finpines, f1, f2)
```

---

kppm *Fit Cluster or Cox Point Process Model*

Description

Fit a homogeneous or inhomogeneous cluster process or Cox point process model to a point pattern.

Usage

```
  kppm(X, ...)

  ## S3 method for class 'formula'
kppm(X,
                clusters = c("Thomas","MatClust","Cauchy","VarGamma","LGCP"),
                ...,
                data=NULL)

  ## S3 method for class 'ppp'
kppm(X,
       trend = ~1,
       clusters = c("Thomas","MatClust","Cauchy","VarGamma","LGCP"),
       data = NULL,
       ...,
       covariates=data,
       subset,
       method = c("mincon", "clik2", "palm"),
       improve.type = c("none", "clik1", "wclik1", "quasi"),
       improve.args = list(),
       weightfun=NULL,
       control=list(),
       algorithm="Nelder-Mead",
       statistic="K",
       statargs=list(),
       rmax = NULL,
       covfunargs=NULL,
```

```
        use.gam=FALSE,
        nd=NULL, eps=NULL)

## S3 method for class 'quad'
kppm(X,
        trend = ~1,
        clusters = c("Thomas","MatClust","Cauchy","VarGamma","LGCP"),
        data = NULL,
        ...,
        covariates=data,
        subset,
        method = c("mincon", "clik2", "palm"),
        improve.type = c("none", "clik1", "wclik1", "quasi"),
        improve.args = list(),
        weightfun=NULL,
        control=list(),
        algorithm="Nelder-Mead",
        statistic="K",
        statargs=list(),
        rmax = NULL,
        covfunargs=NULL,
        use.gam=FALSE,
        nd=NULL, eps=NULL)
```

Arguments

`X`	A point pattern dataset (object of class "ppp" or "quad") to which the model should be fitted, or a `formula` in the R language defining the model. See Details.
`trend`	An R formula, with no left hand side, specifying the form of the log intensity.
`clusters`	Character string determining the cluster model. Partially matched. Options are "Thomas", "MatClust", "Cauchy", "VarGamma" and "LGCP".
`data,covariates`	The values of spatial covariates (other than the Cartesian coordinates) required by the model. A named list of pixel images, functions, windows, tessellations or numeric constants.
`...`	Additional arguments. See Details.
`subset`	Optional. A subset of the spatial domain, to which the model-fitting should be restricted. A window (object of class "owin") or a logical-valued pixel image (object of class "im"), or an expression (possibly involving the names of entries in `data`) which can be evaluated to yield a window or pixel image.
`method`	The fitting method. Either "mincon" for minimum contrast, "clik2" for second order composite likelihood, or "palm" for Palm likelihood. Partially matched.
`improve.type`	Method for updating the initial estimate of the trend. Initially the trend is estimated as if the process is an inhomogeneous Poisson process. The default, `improve.type = "none"`, is to use this initial estimate. Otherwise, the trend estimate is updated by `improve.kppm`, using information about the pair correlation function. Options are "clik1" (first order composite likelihood, essentially equivalent to "none"), "wclik1" (weighted first order composite likelihood) and "quasi" (quasi likelihood).
`improve.args`	Additional arguments passed to `improve.kppm` when `improve.type != "none"`. See Details.

weightfun
: Optional weighting function w in the composite likelihood or Palm likelihood. A function in the R language. See Details.

control
: List of control parameters passed to the optimization function optim.

algorithm
: Character string determining the mathematical optimisation algorithm to be used by optim. See the argument method of optim.

statistic
: Name of the summary statistic to be used for minimum contrast estimation: either "K" or "pcf".

statargs
: Optional list of arguments to be used when calculating the statistic. See Details.

rmax
: Maximum value of interpoint distance to use in the composite likelihood.

covfunargs,use.gam,nd,eps
: Arguments passed to ppm when fitting the intensity.

Details

This function fits a clustered point process model to the point pattern dataset X.

The model may be either a *Neyman-Scott cluster process* or another *Cox process*. The type of model is determined by the argument clusters. Currently the options are clusters="Thomas" for the Thomas process, clusters="MatClust" for the Matern cluster process, clusters="Cauchy" for the Neyman-Scott cluster process with Cauchy kernel, clusters="VarGamma" for the Neyman-Scott cluster process with Variance Gamma kernel (requires an additional argument nu to be passed through the dots; see rVarGamma for details), and clusters="LGCP" for the log-Gaussian Cox process (may require additional arguments passed through ...; see rLGCP for details on argument names). The first four models are Neyman-Scott cluster processes.

The algorithm first estimates the intensity function of the point process using ppm. The argument X may be a point pattern (object of class "ppp") or a quadrature scheme (object of class "quad"). The intensity is specified by the trend argument. If the trend formula is ~1 (the default) then the model is *homogeneous*. The algorithm begins by estimating the intensity as the number of points divided by the area of the window. Otherwise, the model is *inhomogeneous*. The algorithm begins by fitting a Poisson process with log intensity of the form specified by the formula trend. (See ppm for further explanation).

The argument X may also be a formula in the R language. The right hand side of the formula gives the trend as described above. The left hand side of the formula gives the point pattern dataset to which the model should be fitted.

If improve.type="none" this is the final estimate of the intensity. Otherwise, the intensity estimate is updated, as explained in improve.kppm. Additional arguments to improve.kppm are passed as a named list in improve.args.

The clustering parameters of the model are then fitted either by minimum contrast estimation, or by maximum composite likelihood.

Minimum contrast: If method = "mincon" (the default) clustering parameters of the model will be fitted by minimum contrast estimation, that is, by matching the theoretical K-function of the model to the empirical K-function of the data, as explained in mincontrast.

For a homogeneous model (trend = ~1) the empirical K-function of the data is computed using Kest, and the parameters of the cluster model are estimated by the method of minimum contrast.

For an inhomogeneous model, the inhomogeneous K function is estimated by Kinhom using the fitted intensity. Then the parameters of the cluster model are estimated by the method of minimum contrast using the inhomogeneous K function. This two-step estimation procedure is due to Waagepetersen (2007).

If `statistic="pcf"` then instead of using the K-function, the algorithm will use the pair correlation function `pcf` for homogeneous models and the inhomogeneous pair correlation function `pcfinhom` for inhomogeneous models. In this case, the smoothing parameters of the pair correlation can be controlled using the argument `statargs`, as shown in the Examples.

Additional arguments `...` will be passed to `mincontrast` to control the minimum contrast fitting algorithm.

Composite likelihood: If `method = "clik2"` the clustering parameters of the model will be fitted by maximising the second-order composite likelihood (Guan, 2006). The log composite likelihood is

$$\sum_{i,j} w(d_{ij}) \log \rho(d_{ij}; \theta) - \left(\sum_{i,j} w(d_{ij}) \right) \log \int_D \int_D w(\|u - v\|) \rho(\|u - v\|; \theta) \, du \, dv$$

where the sums are taken over all pairs of data points x_i, x_j separated by a distance $d_{ij} = \|x_i - x_j\|$ less than `rmax`, and the double integral is taken over all pairs of locations u, v in the spatial window of the data. Here $\rho(d; \theta)$ is the pair correlation function of the model with cluster parameters θ.

The function w in the composite likelihood is a weighting function and may be chosen arbitrarily. It is specified by the argument `weightfun`. If this is missing or `NULL` then the default is a threshold weight function, $w(d) = 1(d \leq R)$, where R is `rmax/2`.

Palm likelihood: If `method = "palm"` the clustering parameters of the model will be fitted by maximising the Palm loglikelihood (Tanaka et al, 2008)

$$\sum_{i,j} w(x_i, x_j) \log \lambda_P(x_j \mid x_i; \theta) - \int_D w(x_i, u) \lambda_P(u \mid x_i; \theta) \mathrm{d}u$$

with the same notation as above. Here $\lambda_P(u|v; \theta$ is the Palm intensity of the model at location u given there is a point at v.

In all three methods, the optimisation is performed by the generic optimisation algorithm `optim`. The behaviour of this algorithm can be modified using the argument `control`. Useful control arguments include `trace`, `maxit` and `abstol` (documented in the help for `optim`).

Fitting the LGCP model requires the **RandomFields** package, except in the default case where the exponential covariance is assumed.

Value

An object of class `"kppm"` representing the fitted model. There are methods for printing, plotting, predicting, simulating and updating objects of this class.

Log-Gaussian Cox Models

To fit a log-Gaussian Cox model with non-exponential covariance, specify `clusters="LGCP"` and use additional arguments to specify the covariance structure. These additional arguments can be given individually in the call to `kppm`, or they can be collected together in a list called `covmodel`.

For example a Matern model with parameter $\nu = 0.5$ could be specified either by `kppm(X, clusters="LGCP", model="` or by `kppm(X, clusters="LGCP", covmodel=list(model="matern", nu=0.5))`.

The argument `model` specifies the type of covariance model: the default is `model="exp"` for an exponential covariance. Alternatives include `"matern"`, `"cauchy"` and `"spheric"`. Model names correspond to functions beginning with `RM` in the **RandomFields** package: for example `model="matern"` corresponds to the function `RMmatern` in the **RandomFields** package.

Additional arguments are passed to the relevant function in the **RandomFields** package: for example if `model="matern"` then the additional argument `nu` is required, and is passed to the function `RMmatern` in the **RandomFields** package.

Note that it is not possible to use *anisotropic* covariance models because the `kppm` technique assumes the pair correlation function is isotropic.

Error and warning messages

See `ppm.ppp` for a list of common error messages and warnings originating from the first stage of model-fitting.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>, with contributions from Abdollah Jalilian and Rasmus Waagepetersen.

References

Guan, Y. (2006) A composite likelihood approach in fitting spatial point process models. *Journal of the American Statistical Association* **101**, 1502–1512.

Jalilian, A., Guan, Y. and Waagepetersen, R. (2012) Decomposition of variance for spatial Cox processes. *Scandinavian Journal of Statistics* **40**, 119–137.

Tanaka, U. and Ogata, Y. and Stoyan, D. (2008) Parameter estimation and model selection for Neyman-Scott point processes. *Biometrical Journal* **50**, 43–57.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

Methods for `kppm` objects: `plot.kppm`, `fitted.kppm`, `predict.kppm`, `simulate.kppm`, `update.kppm`, `vcov.kppm`, `methods.kppm`, `as.ppm.kppm`, `Kmodel.kppm`, `pcfmodel.kppm`.

Minimum contrast fitting algorithm: `mincontrast`.

Alternative fitting algorithms: `thomas.estK`, `matclust.estK`, `lgcp.estK`, `cauchy.estK`, `vargamma.estK`, `thomas.estpcf`, `matclust.estpcf`, `lgcp.estpcf`, `cauchy.estpcf`, `vargamma.estpcf`,

Summary statistics: `Kest`, `Kinhom`, `pcf`, `pcfinhom`.

See also `ppm`

Examples

```
# method for point patterns
kppm(redwood, ~1, "Thomas")
# method for formulas
kppm(redwood ~ 1, "Thomas")

kppm(redwood ~ 1, "Thomas", method="c")
kppm(redwood ~ 1, "Thomas", method="p")

kppm(redwood ~ x, "MatClust")
kppm(redwood ~ x, "MatClust", statistic="pcf", statargs=list(stoyan=0.2))
kppm(redwood ~ x, cluster="Cauchy", statistic="K")
kppm(redwood, cluster="VarGamma", nu = 0.5, statistic="pcf")
```

```
# LGCP models
kppm(redwood ~ 1, "LGCP", statistic="pcf")
if(require("RandomFields")) {
  kppm(redwood ~ x, "LGCP", statistic="pcf",
                            model="matern", nu=0.3,
                            control=list(maxit=10))
}

# fit with composite likelihood method
kppm(redwood ~ x, "VarGamma", method="clik2", nu.ker=-3/8)

# fit intensity with quasi-likelihood method
kppm(redwood ~ x, "Thomas", improve.type = "quasi")
```

Kres *Residual K Function*

Description

Given a point process model fitted to a point pattern dataset, this function computes the residual K function, which serves as a diagnostic for goodness-of-fit of the model.

Usage

```
Kres(object, ...)
```

Arguments

object	Object to be analysed. Either a fitted point process model (object of class "ppm"), a point pattern (object of class "ppp"), a quadrature scheme (object of class "quad"), or the value returned by a previous call to Kcom.
...	Arguments passed to Kcom.

Details

This command provides a diagnostic for the goodness-of-fit of a point process model fitted to a point pattern dataset. It computes a residual version of the K function of the dataset, which should be approximately zero if the model is a good fit to the data.

In normal use, object is a fitted point process model or a point pattern. Then Kres first calls Kcom to compute both the nonparametric estimate of the K function and its model compensator. Then Kres computes the difference between them, which is the residual K-function.

Alternatively, object may be a function value table (object of class "fv") that was returned by a previous call to Kcom. Then Kres computes the residual from this object.

Value

A function value table (object of class "fv"), essentially a data frame of function values. There is a plot method for this class. See fv.object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Ege Rubak <rubak@math.aau.dk> and Jesper Møller.

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Related functions: Kcom, Kest.

Alternative functions: Gres, psstG, psstA, psst.

Point process models: ppm.

Examples

```
    data(cells)
    fit0 <- ppm(cells, ~1) # uniform Poisson

    K0 <- Kres(fit0)
    K0
    plot(K0)
# isotropic-correction estimate
    plot(K0, ires ~ r)
# uniform Poisson is clearly not correct

    fit1 <- ppm(cells, ~1, Strauss(0.08))

    K1 <- Kres(fit1)

    if(interactive()) {
      plot(K1, ires ~ r)
   # fit looks approximately OK; try adjusting interaction distance
      plot(Kres(cells, interaction=Strauss(0.12)))
    }

# How to make envelopes
    ## Not run:
    E <- envelope(fit1, Kres, model=fit1, nsim=19)
    plot(E)

## End(Not run)

# For computational efficiency
    Kc <- Kcom(fit1)
    K1 <- Kres(Kc)
```

Kscaled *Locally Scaled K-function*

Description

Estimates the locally-rescaled K-function of a point process.

Usage

```
Kscaled(X, lambda=NULL, ..., r = NULL, breaks = NULL,
  rmax = 2.5,
  correction=c("border", "isotropic", "translate"),
  renormalise=FALSE, normpower=1,
  sigma=NULL, varcov=NULL)

Lscaled(...)
```

Arguments

`X`	The observed data point pattern, from which an estimate of the locally scaled K function will be computed. An object of class "ppp" or in a format recognised by `as.ppp()`.
`lambda`	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern `X`, a pixel image (object of class "im") giving the intensity values at all locations, a `function(x,y)` which can be evaluated to give the intensity value at any location, or a fitted point process model (object of class "ppm").
`...`	Arguments passed from `Lscaled` to `Kscaled` and from `Kscaled` to `density.ppp` if `lambda` is omitted.
`r`	vector of values for the argument r at which the locally scaled K function should be evaluated. (These are rescaled distances.) Not normally given by the user; there is a sensible default.
`breaks`	This argument is for internal use only.
`rmax`	maximum value of the argument r that should be used. (This is the rescaled distance).
`correction`	A character vector containing any selection of the options "border", "isotropic", "Ripley", "translate", "translation", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively `correction="all"` selects all options.
`renormalise`	Logical. Whether to renormalise the estimate. See Details.
`normpower`	Integer (usually either 1 or 2). Normalisation power. See Details.
`sigma,varcov`	Optional arguments passed to `density.ppp` to control the smoothing bandwidth, when `lambda` is estimated by kernel smoothing.

Details

Kscaled computes an estimate of the K function for a locally scaled point process. Lscaled computes the corresponding L function $L(r) = \sqrt{K(r)/\pi}$.

Locally scaled point processes are a class of models for inhomogeneous point patterns, introduced by Hahn et al (2003). They include inhomogeneous Poisson processes, and many other models.

The template K function of a locally-scaled process is a counterpart of the "ordinary" Ripley K function, in which the distances between points of the process are measured on a spatially-varying scale (such that the locally rescaled process has unit intensity).

The template K function is an indicator of interaction between the points. For an inhomogeneous Poisson process, the theoretical template K function is approximately equal to $K(r) = \pi r^2$. Values $K_{\mathrm{scaled}}(r) > \pi r^2$ are suggestive of clustering.

Kscaled computes an estimate of the template K function and Lscaled computes the corresponding L function $L(r) = \sqrt{K(r)/\pi}$.

The locally scaled interpoint distances are computed using an approximation proposed by Hahn (2007). The Euclidean distance between two points is multiplied by the average of the square roots of the intensity values at the two points.

The argument lambda should supply the (estimated) values of the intensity function λ. It may be either

a numeric vector containing the values of the intensity function at the points of the pattern X.

a pixel image (object of class "im") assumed to contain the values of the intensity function at all locations in the window.

a function which can be evaluated to give values of the intensity at any locations.

omitted: if lambda is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

If lambda is a numeric vector, then its length should be equal to the number of points in the pattern X. The value lambda[i] is assumed to be the the (estimated) value of the intensity $\lambda(x_i)$ for the point x_i of the pattern X. Each value must be a positive number; NA's are not allowed.

If lambda is a pixel image, the domain of the image should cover the entire window of the point pattern. If it does not (which may occur near the boundary because of discretisation error), then the missing pixel values will be obtained by applying a Gaussian blur to lambda using blur, then looking up the values of this blurred image for the missing locations. (A warning will be issued in this case.)

If lambda is a function, then it will be evaluated in the form lambda(x,y) where x and y are vectors of coordinates of the points of X. It should return a numeric vector with length equal to the number of points in X.

If lambda is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate lambda[i] for the point X[i] is computed by removing X[i] from the point pattern, applying kernel smoothing to the remaining points using density.ppp, and evaluating the smoothed intensity at the point X[i]. The smoothing kernel bandwidth is controlled by the arguments sigma and varcov, which are passed to density.ppp along with any extra arguments.

If renormalise=TRUE, the estimated intensity lambda is multiplied by $c^(normpower/2)$ before performing other calculations, where $c = area(W)/sum[i](1/lambda(x[i]))$. This renormalisation has about the same effect as in Kinhom, reducing the variability and bias of the estimate in small samples and in cases of very strong inhomogeneity.

Edge corrections are used to correct bias in the estimation of K_{scaled}. First the interpoint distances are rescaled, and then edge corrections are applied as in Kest. See Kest for details of the edge corrections and the options for the argument correction.

The pair correlation function can also be applied to the result of `Kscaled`; see `pcf` and `pcf.fv`.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing at least the following columns,

- `r` — the vector of values of the argument r at which the pair correlation function $g(r)$ has been estimated
- `theo` — vector of values of πr^2, the theoretical value of $K_{\text{scaled}}(r)$ for an inhomogeneous Poisson process

and containing additional columns according to the choice specified in the `correction` argument. The additional columns are named `border`, `trans` and `iso` and give the estimated values of $K_{\text{scaled}}(r)$ using the border correction, translation correction, and Ripley isotropic correction, respectively.

Author(s)

Ute Hahn, Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Møller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

Hahn, U. (2007) *Global and Local Scaling in the Statistics of Spatial Point Processes*. Habilitationsschrift, Universitaet Augsburg.

Hahn, U., Jensen, E.B.V., van Lieshout, M.N.M. and Nielsen, L.S. (2003) Inhomogeneous spatial point processes by location-dependent scaling. *Advances in Applied Probability* **35**, 319–336.

Prokešová, M., Hahn, U. and Vedel Jensen, E.B. (2006) Statistics for locally scaled point patterns. In A. Baddeley, P. Gregori, J. Mateu, R. Stoica and D. Stoyan (eds.) *Case Studies in Spatial Point Pattern Modelling*. Lecture Notes in Statistics 185. New York: Springer Verlag. Pages 99–123.

See Also

`Kest`, `pcf`

Examples

```
data(bronzefilter)
X <- unmark(bronzefilter)
K <- Kscaled(X)
fit <- ppm(X, ~x)
lam <- predict(fit)
K <- Kscaled(X, lam)
```

Ksector	*Sector K-function*

Description

A directional counterpart of Ripley's K function, in which pairs of points are counted only when the vector joining the pair happens to lie in a particular range of angles.

Usage

```
Ksector(X, begin = 0, end = 360, ...,
        units = c("degrees", "radians"),
        r = NULL, breaks = NULL,
        correction = c("border", "isotropic", "Ripley", "translate"),
        domain=NULL, ratio = FALSE, verbose=TRUE)
```

Arguments

`X`	The observed point pattern, from which an estimate of $K(r)$ will be computed. An object of class `"ppp"`, or data in any format acceptable to `as.ppp()`.
`begin,end`	Numeric values giving the range of angles inside which points will be counted. Angles are measured in degrees (if `units="degrees"`, the default) or radians (if `units="radians"`) anti-clockwise from the positive x-axis.
`...`	Ignored.
`units`	Units in which the angles `begin` and `end` are expressed.
`r`	Optional. Vector of values for the argument r at which $K(r)$ should be evaluated. Users are advised *not* to specify this argument; there is a sensible default.
`breaks`	This argument is for internal use only.
`correction`	Optional. A character vector containing any selection of the options `"none"`, `"border"`, `"bord.modif"`, `"isotropic"`, `"Ripley"`, `"translate"`, `"translation"`, `"none"`, `"good"` or `"best"`. It specifies the edge correction(s) to be applied. Alternatively `correction="all"` selects all options.
`domain`	Optional window. The first point x_i of each pair of points will be constrained to lie in `domain`.
`ratio`	Logical. If `TRUE`, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
`verbose`	Logical value indicating whether to print progress reports and warnings.

Details

This is a directional counterpart of Ripley's K function (see `Kest`) in which, instead of counting all pairs of points within a specified distance r, we count only the pairs (x_i, x_j) for which the vector $x_j - x_i$ falls in a particular range of angles.

This can be used to evaluate evidence for anisotropy in the point pattern `X`.

Value

An object of class `"fv"` containing the estimated function.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

Kest

Examples

```
K <- Ksector(swedishpines, 0, 90)
plot(K)
```

LambertW *Lambert's W Function*

Description

Computes Lambert's W-function.

Usage

```
LambertW(x)
```

Arguments

x	Vector of nonnegative numbers.

Details

Lambert's W-function is the inverse function of $f(y) = ye^y$. That is, W is the function such that

$$W(x)e^{W(x)} = x$$

This command LambertW computes $W(x)$ for each entry in the argument x. If the library **gsl** has been installed, then the function lambert_W0 in that library is invoked. Otherwise, values of the W-function are computed by root-finding, using the function uniroot.

Computation using **gsl** is about 100 times faster.

If any entries of x are infinite or NA, the corresponding results are NA.

Value

Numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Corless, R, Gonnet, G, Hare, D, Jeffrey, D and Knuth, D (1996), On the Lambert W function. *Computational Mathematics*, **5**, 325–359.

Roy, R and Olver, F (2010), Lambert W function. In Olver, F, Lozier, D and Boisvert, R (eds.), *NIST Handbook of Mathematical Functions*, Cambridge University Press.

Examples

```
LambertW(exp(1))
```

`laslett` *Laslett's Transform*

Description

Apply Laslett's Transform to a spatial region, returning the original and transformed regions, and the original and transformed positions of the lower tangent points. This is a diagnostic for the Boolean model.

Usage

```
laslett(X, ..., verbose = FALSE, plotit = TRUE, discretise = FALSE,
        type=c("lower", "upper", "left", "right"))
```

Arguments

`X`	Spatial region to be transformed. A window (object of class `"owin"`) or a logical-valued pixel image (object of class `"im"`).
`...`	Graphics arguments to control the plot (passed to `plot.laslett` when `plotit=TRUE`) or arguments determining the pixel resolution (passed to `as.mask`).
`verbose`	Logical value indicating whether to print progress reports.
`plotit`	Logical value indicating whether to plot the result.
`discretise`	Logical value indicating whether polygonal windows should first be converted to pixel masks before the Laslett transform is computed. This should be set to `TRUE` for very complicated polygons.
`type`	Type of tangent points to be detected. This also determines the direction of contraction in the set transformation. Default is `type="lower"`.

Details

This function finds the lower tangent points of the spatial region `X`, then applies Laslett's Transform to the space, and records the transformed positions of the lower tangent points.

Laslett's transform is a diagnostic for the Boolean Model. A test of the Boolean model can be performed by applying a test of CSR to the transformed tangent points. See the Examples.

The rationale is that, if the region `X` was generated by a Boolean model with convex grains, then the lower tangent points of `X`, when subjected to Laslett's transform, become a Poisson point process (Cressie, 1993, section 9.3.5; Molchanov, 1997; Barbour and Schmidt, 2001).

Intuitively, Laslett's transform is a way to account for the fact that tangent points of `X` cannot occur *inside* `X`. It treats the interior of `X` as empty space, and collapses this empty space so that only the *exterior* of `X` remains. In this collapsed space, the tangent points are completely random.

Formally, Laslett's transform is a random (i.e. data-dependent) spatial transformation which maps each spatial location (x, y) to a new location (x', y) at the same height y. The transformation is defined so that x' is the total *uncovered* length of the line segment from $(0, y)$ to (x, y), that is, the total length of the parts of this segment that fall outside the region `X`.

In more colourful terms, suppose we use an abacus to display a pixellated version of `X`. Each wire of the abacus represents one horizontal line in the pixel image. Each pixel lying *outside* the region `X` is represented by a bead of the abacus; pixels *inside* `X` are represented by the absence of a bead. Next we find any beads which are lower tangent points of `X`, and paint them green. Then Laslett's Transform is applied by pushing all beads to the left, as far as possible. The final locations of all the beads provide a new spatial region, inside which is the point pattern of tangent points (marked by the green-painted beads).

If `plotit=TRUE` (the default), a before-and-after plot is generated, showing the region `X` and the tangent points before and after the transformation. This plot can also be generated by calling `plot(a)` where `a` is the object returned by the function `laslett`.

If the argument `type` is given, then this determines the type of tangents that will be detected, and also the direction of contraction in Laslett's transform. The computation is performed by first rotating `X`, applying Laslett's transform for lower tangent points, then rotating back.

There are separate algorithms for polygonal windows and pixellated windows (binary masks). The polygonal algorithm may be slow for very complicated polygons. If this happens, setting `discretise=TRUE` will convert the polygonal window to a binary mask and invoke the pixel raster algorithm.

Value

A list, which also belongs to the class `"laslett"` so that it can immediately be printed and plotted.

The list elements are:

oldX: the original dataset `X`;

TanOld: a point pattern, whose window is `Frame(X)`, containing the lower tangent points of `X`;

TanNew: a point pattern, whose window is the Laslett transform of `Frame(X)`, and which contains the Laslett-transformed positions of the tangent points;

Rect: a rectangular window, which is the largest rectangle lying inside the transformed set;

df: a data frame giving the locations of the tangent points before and after transformation.

type: character string specifying the type of tangents.

Author(s)

Kassel Hingee and Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`.

References

Barbour, A.D. and Schmidt, V. (2001) On Laslett's Transform for the Boolean Model. *Advances in Applied Probability* **33**(1), 1–5.

Cressie, N.A.C. (1993) *Statistics for spatial data*, second edition. John Wiley and Sons.

Molchanov, I. (1997) *Statistics of the Boolean Model for Practitioners and Mathematicians*. Wiley.

See Also

plot.laslett

Examples

```
a <- laslett(heather$coarse)
with(a, clarkevans.test(TanNew[Rect], correction="D", nsim=39))
X <- discs(runifpoint(15) %mark% 0.2, npoly=16)
b <- laslett(X)
```

---

latest.news *Print News About Latest Version of Package*

Description

Prints the news documentation for the current version of spatstat or another specified package.

Usage

```
latest.news(package = "spatstat", doBrowse=FALSE)
```

Arguments

package	Name of package for which the latest news should be printed.
doBrowse	Logical value indicating whether to display the results in a browser window instead of printing them.

Details

By default, this function prints the news documentation about changes in the current installed version of the **spatstat** package. The function can be called simply by typing its name without parentheses (see the Examples).

If package is given, then the function reads the news for the specified package from its NEWS file (if it has one) and prints only the entries that refer to the current version of the package.

To see the news for all previous versions as well as the current version, use the R utility news. See the Examples.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

news, bugfixes

Examples

```
if(interactive()) {

  # current news
  latest.news

  # all news
  news(package="spatstat")

}
```

---

layered — *Create List of Plotting Layers*

Description

Given several objects which are capable of being plotted, create a list containing these objects as if they were successive layers of a plot. The list can then be plotted in different ways.

Usage

```
layered(..., plotargs = NULL, LayerList=NULL)
```

Arguments

`...`	Objects which can be plotted by `plot`.
`plotargs`	Default values of the plotting arguments for each of the objects. A list of lists of arguments of the form `name=value`.
`LayerList`	A list of objects. Incompatible with `...`.

Details

Layering is a simple mechanism for controlling a high-level plot that is composed of several successive plots, for example, a background and a foreground plot. The layering mechanism makes it easier to issue the plot command, to switch on or off the plotting of each individual layer, to control the plotting arguments that are passed to each layer, and to zoom in.

Each individual layer in the plot should be saved as an object that can be plotted using `plot`. It will typically belong to some class, which has a method for the generic function `plot`.

The command `layered` simply saves the objects `...` as a list of class `"layered"`. This list can then be plotted by the method `plot.layered`. Thus, you only need to type a single `plot` command to produce the multi-layered plot. Individual layers of the plot can be switched on or off, or manipulated, using arguments to `plot.layered`.

The argument `plotargs` contains default values of the plotting arguments for each layer. It should be a list, with one entry for each object in `...`. Each entry of `plotargs` should be a list of arguments in the form `name=value`, which are recognised by the `plot` method for the relevant layer.

The `plotargs` can also include an argument named `.plot` specifying (the name of) a function to perform the plotting instead of the generic `plot`.

The length of `plotargs` should either be equal to the number of layers, or equal to 1. In the latter case it will be replicated to the appropriate length.

Value

A list, belonging to the class "layered". There are methods for plot, "[", "shift", "affine", "rotate" and "rescale".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

plot.layered, methods.layered, as.layered, [.layered, layerplotargs.

Examples

```
D <- distmap(cells)
L <- layered(D, cells)
L
L <- layered(D, cells,
 plotargs=list(list(ribbon=FALSE), list(pch=16)))
plot(L)

layerplotargs(L)[[1]] <- list(.plot="contour")
plot(L)
```

---

layerplotargs *Extract or Replace the Plot Arguments of a Layered Object*

Description

Extracts or replaces the plot arguments of a layered object.

Usage

```
layerplotargs(L)

layerplotargs(L) <- value
```

Arguments

L	An object of class "layered" created by the function layered.
value	Replacement value. A list, with the same length as L, whose elements are lists of plot arguments.

Details

These commands extract or replace the plotargs in a layered object. See layered.

The replacement value should normally have the same length as the current value. However, it can also be a list with *one* element which is a list of parameters. This will be replicated to the required length.

For the assignment function layerplotargs<-, the argument L can be any spatial object; it will be converted to a layered object with a single layer.

Value

layerplotargs returns a list of lists of plot arguments.

"layerplotargs<-" returns the updated object of class "layered".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

layered, methods.layered, [.layered.

Examples

```
W <- square(2)
L <- layered(W=W, X=cells)
## The following are equivalent
layerplotargs(L) <- list(list(), list(pch=16))
layerplotargs(L)[[2]] <- list(pch=16)
layerplotargs(L)$X <- list(pch=16)

## The following are equivalent
layerplotargs(L) <- list(list(cex=2), list(cex=2))
layerplotargs(L) <- list(list(cex=2))
```

---

layout.boxes *Generate a Row or Column Arrangement of Rectangles.*

Description

A simple utility to generate a row or column of boxes (rectangles) for use in point-and-click panels.

Usage

```
layout.boxes(B, n, horizontal = FALSE, aspect = 0.5, usefrac = 0.9)
```

Arguments

B	Bounding rectangle for the boxes. An object of class "owin".
n	Integer. The number of boxes.
horizontal	Logical. If TRUE, arrange the boxes in a horizontal row. If FALSE (the default), arrange them in a vertical column.
aspect	Aspect ratio (height/width) of each box.
usefrac	Number between 0 and 1. The fraction of height or width of B that should be occupied by boxes.

Details

This simple utility generates a list of boxes (rectangles) inside the bounding box B arranged in a regular row or column. It is useful for generating the positions of the panel buttons in the function simplepanel.

Value

A list of rectangles.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

simplepanel

Examples

```
B <- owin(c(0,10),c(0,1))
boxes <- layout.boxes(B, 5, horizontal=TRUE)
plot(B, main="", col="blue")
niets <- lapply(boxes, plot, add=TRUE, col="grey")
```

Lcross *Multitype L-function (cross-type)*

Description

Calculates an estimate of the cross-type L-function for a multitype point pattern.

Usage

```
Lcross(X, i, j, ..., from, to)
```

Arguments

X	The observed point pattern, from which an estimate of the cross-type L function $L_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
...	Arguments passed to Kcross.
from,to	An alternative way to specify i and j respectively.

Details

The cross-type L-function is a transformation of the cross-type K-function,

$$L_{ij}(r) = \sqrt{\frac{K_{ij}(r)}{\pi}}$$

where $K_{ij}(r)$ is the cross-type K-function from type `i` to type `j`. See `Kcross` for information about the cross-type K-function.

The command `Lcross` first calls `Kcross` to compute the estimate of the cross-type K-function, and then applies the square root transformation.

For a marked point pattern in which the points of type `i` are independent of the points of type `j`, the theoretical value of the L-function is $L_{ij}(r) = r$. The square root also has the effect of stabilising the variance of the estimator, so that L_{ij} is more appropriate for use in simulation envelopes and hypothesis tests.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing columns

`r`	the vector of values of the argument r at which the function L_{ij} has been estimated
`theo`	the theoretical value $L_{ij}(r) = r$ for a stationary Poisson process

together with columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function L_{ij} obtained by the edge corrections named.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`Kcross`, `Ldot`, `Lest`

Examples

```
data(amacrine)
L <- Lcross(amacrine, "off", "on")
plot(L)
```

Lcross.inhom *Inhomogeneous Cross Type L Function*

Description

For a multitype point pattern, estimate the inhomogeneous version of the cross-type L function.

Usage

```
Lcross.inhom(X, i, j, ...)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous cross type L function $L_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of `marks(X)`.
...	Other arguments passed to `Kcross.inhom`.

Details

This is a generalisation of the function `Lcross` to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function `Linhom`.

All the arguments are passed to `Kcross.inhom`, which estimates the inhomogeneous multitype K function $K_{ij}(r)$ for the point pattern. The resulting values are then transformed by taking $L(r) = \sqrt{K(r)/\pi}$.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $L_{ij}(r)$ has been estimated
theo	the theoretical value of $L_{ij}(r)$ for a marked Poisson process, identically equal to r

together with a column or columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $L_{ij}(r)$ obtained by the edge corrections named.

Warnings

The arguments `i` and `j` are always interpreted as levels of the factor `X$marks`. They are converted to character strings if they are not already character strings. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

References

Møller, J. and Waagepetersen, R. Statistical Inference and Simulation for Spatial Point Processes Chapman and Hall/CRC Boca Raton, 2003.

See Also

Lcross, Linhom, Kcross.inhom

Examples

```
# Lansing Woods data
woods <- lansing

ma <- split(woods)$maple
wh <- split(woods)$whiteoak

# method (1): estimate intensities by nonparametric smoothing
lambdaM <- density.ppp(ma, sigma=0.15, at="points")
lambdaW <- density.ppp(wh, sigma=0.15, at="points")
L <- Lcross.inhom(woods, "whiteoak", "maple", lambdaW, lambdaM)

# method (2): fit parametric intensity model
fit <- ppm(woods ~marks * polynom(x,y,2))
# evaluate fitted intensities at data points
# (these are the intensities of the sub-processes of each type)
inten <- fitted(fit, dataonly=TRUE)
# split according to types of points
lambda <- split(inten, marks(woods))
L <- Lcross.inhom(woods, "whiteoak", "maple",
          lambda$whiteoak, lambda$maple)

# synthetic example: type A points have intensity 50,
#                    type B points have intensity 100 * x
lamB <- as.im(function(x,y){50 + 100 * x}, owin())
X <- superimpose(A=runifpoispp(50), B=rpoispp(lamB))
L <- Lcross.inhom(X, "A", "B",
    lambdaI=as.im(50, Window(X)), lambdaJ=lamB)
```

Ldot *Multitype L-function (i-to-any)*

Description

Calculates an estimate of the multitype L-function (from type i to any type) for a multitype point pattern.

Usage

```
Ldot(X, i, ..., from)
```

Arguments

X	The observed point pattern, from which an estimate of the dot-type L function $L_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
...	Arguments passed to Kdot.
from	An alternative way to specify i.

Details

This command computes

$$L_{i\bullet}(r) = \sqrt{\frac{K_{i\bullet}(r)}{\pi}}$$

where $K_{i\bullet}(r)$ is the multitype K-function from points of type i to points of any type. See Kdot for information about $K_{i\bullet}(r)$.

The command Ldot first calls Kdot to compute the estimate of the i-to-any K-function, and then applies the square root transformation.

For a marked Poisson point process, the theoretical value of the L-function is $L_{i\bullet}(r) = r$. The square root also has the effect of stabilising the variance of the estimator, so that $L_{i\bullet}$ is more appropriate for use in simulation envelopes and hypothesis tests.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Essentially a data frame containing columns

r	the vector of values of the argument r at which the function $L_{i\bullet}$ has been estimated
theo	the theoretical value $L_{i\bullet}(r) = r$ for a stationary Poisson process

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $L_{i\bullet}$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Kdot, Lcross, Lest

Examples

```
data(amacrine)
L <- Ldot(amacrine, "off")
plot(L)
```

`Ldot.inhom` *Inhomogeneous Multitype L Dot Function*

Description

For a multitype point pattern, estimate the inhomogeneous version of the dot L function.

Usage

```
Ldot.inhom(X, i, ...)
```

Arguments

`X`	The observed point pattern, from which an estimate of the inhomogeneous cross type L function $L_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
`i`	The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.
`...`	Other arguments passed to `Kdot.inhom`.

Details

This a generalisation of the function `Ldot` to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function `Linhom`.

All the arguments are passed to `Kdot.inhom`, which estimates the inhomogeneous multitype K function $K_{i\bullet}(r)$ for the point pattern. The resulting values are then transformed by taking $L(r) = \sqrt{K(r)/\pi}$.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing numeric columns

`r`	the values of the argument r at which the function $L_{i\bullet}(r)$ has been estimated
`theo`	the theoretical value of $L_{i\bullet}(r)$ for a marked Poisson process, identical to r.

together with a column or columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $L_{i\bullet}(r)$ obtained by the edge corrections named.

Warnings

The argument `i` is interpreted as a level of the factor `X$marks`. It is converted to a character string if it is not already a character string. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

References

Møller, J. and Waagepetersen, R. Statistical Inference and Simulation for Spatial Point Processes Chapman and Hall/CRC Boca Raton, 2003.

See Also

Ldot, Linhom, Kdot.inhom, Lcross.inhom.

Examples

```
# Lansing Woods data
lan <- lansing
lan <- lan[seq(1,npoints(lan), by=10)]
ma <- split(lan)$maple
lg <- unmark(lan)

# Estimate intensities by nonparametric smoothing
lambdaM <- density.ppp(ma, sigma=0.15, at="points")
lambdadot <- density.ppp(lg, sigma=0.15, at="points")
L <- Ldot.inhom(lan, "maple", lambdaI=lambdaM,
                              lambdadot=lambdadot)


# synthetic example: type A points have intensity 50,
#                    type B points have intensity 50 + 100 * x
lamB <- as.im(function(x,y){50 + 100 * x}, owin())
lamdot <- as.im(function(x,y) { 100 + 100 * x}, owin())
X <- superimpose(A=runifpoispp(50), B=rpoispp(lamB))
L <- Ldot.inhom(X, "B",  lambdaI=lamB,     lambdadot=lamdot)
```

`lengths.psp` *Lengths of Line Segments*

Description

Computes the length of each line segment in a line segment pattern.

Usage

```
lengths.psp(x, squared=FALSE)
```

Arguments

`x`	A line segment pattern (object of class "psp").
`squared`	Logical value indicating whether to return the squared lengths (`squared=TRUE`) or the lengths themselves (`squared=FALSE`, the default).

Details

The length of each line segment is computed and the lengths are returned as a numeric vector.

Using squared lengths may be more efficient for some purposes, for example, to find the length of the shortest segment, `sqrt(min(lengths.psp(x, squared=TRUE)))` is faster than `min(lengths.psp(x))`.

Value

Numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

summary.psp, midpoints.psp, angles.psp

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
b <- lengths.psp(a)
```

LennardJones *The Lennard-Jones Potential*

Description

Creates the Lennard-Jones pairwise interaction structure which can then be fitted to point pattern data.

Usage

```
LennardJones(sigma0=NA)
```

Arguments

sigma0	Optional. Initial estimate of the parameter σ. A positive number.

Details

In a pairwise interaction point process with the Lennard-Jones pair potential (Lennard-Jones, 1924) each pair of points in the point pattern, a distance d apart, contributes a factor

$$v(d) = \exp\left\{-4\epsilon\left[\left(\frac{\sigma}{d}\right)^{12} - \left(\frac{\sigma}{d}\right)^{6}\right]\right\}$$

to the probability density, where σ and ϵ are positive parameters to be estimated.

See **Examples** for a plot of this expression.

This potential causes very strong inhibition between points at short range, and attraction between points at medium range. The parameter σ is called the *characteristic diameter* and controls the scale of interaction. The parameter ϵ is called the *well depth* and determines the strength of attraction. The potential switches from inhibition to attraction at $d = \sigma$. The maximum value of the pair potential is $\exp(\epsilon)$ occuring at distance $d = 2^{1/6}\sigma$. Interaction is usually considered to be negligible for distances $d > 2.5\sigma\max\{1, \epsilon^{1/6}\}$.

This potential is used to model interactions between uncharged molecules in statistical physics.

The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Lennard-Jones pairwise interaction is yielded by the function LennardJones(). See the examples below.

Value

An object of class "interact" describing the Lennard-Jones interpoint interaction structure.

Rescaling

To avoid numerical instability, the interpoint distances d are rescaled when fitting the model.

Distances are rescaled by dividing by sigma0. In the formula for $v(d)$ above, the interpoint distance d will be replaced by d/sigma0.

The rescaling happens automatically by default. If the argument sigma0 is missing or NA (the default), then sigma0 is taken to be the minimum nearest-neighbour distance in the data point pattern (in the call to ppm).

If the argument sigma0 is given, it should be a positive number, and it should be a rough estimate of the parameter σ.

The "canonical regular parameters" estimated by ppm are $\theta_1 = 4\epsilon(\sigma/\sigma_0)^{12}$ and $\theta_2 = 4\epsilon(\sigma/\sigma_0)^6$.

Warnings and Errors

Fitting the Lennard-Jones model is extremely unstable, because of the strong dependence between the functions d^{-12} and d^{-6}. The fitting algorithm often fails to converge. Try increasing the number of iterations of the GLM fitting algorithm, by setting gcontrol=list(maxit=1e3) in the call to ppm.

Errors are likely to occur if this model is fitted to a point pattern dataset which does not exhibit both short-range inhibition and medium-range attraction between points. The values of the parameters σ and ϵ may be NA (because the fitted canonical parameters have opposite sign, which usually occurs when the pattern is completely random).

An absence of warnings does not mean that the fitted model is sensible. A negative value of ϵ may be obtained (usually when the pattern is strongly clustered); this does not correspond to a valid point process model, but the software does not issue a warning.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Lennard-Jones, J.E. (1924) On the determination of molecular fields. *Proc Royal Soc London A* **106**, 463–477.

See Also

ppm, pairwise.family, ppm.object

Examples

```
fit <- ppm(cells ~1, LennardJones(), rbord=0.1)
fit
plot(fitin(fit))
```

Lest *L-function*

Description

Calculates an estimate of the L-function (Besag's transformation of Ripley's K-function) for a spatial point pattern.

Usage

```
Lest(X, ...)
```

Arguments

`X`	The observed point pattern, from which an estimate of $L(r)$ will be computed. An object of class `"ppp"`, or data in any format acceptable to `as.ppp()`.
`...`	Other arguments passed to `Kest` to control the estimation procedure.

Details

This command computes an estimate of the L-function for the spatial point pattern `X`. The L-function is a transformation of Ripley's K-function,

$$L(r) = \sqrt{\frac{K(r)}{\pi}}$$

where $K(r)$ is the K-function.

See `Kest` for information about Ripley's K-function. The transformation to L was proposed by Besag (1977).

The command `Lest` first calls `Kest` to compute the estimate of the K-function, and then applies the square root transformation.

For a completely random (uniform Poisson) point pattern, the theoretical value of the L-function is $L(r) = r$. The square root also has the effect of stabilising the variance of the estimator, so that $L(r)$ is more appropriate for use in simulation envelopes and hypothesis tests.

See `Kest` for the list of arguments.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing columns

`r`	the vector of values of the argument r at which the function L has been estimated
`theo`	the theoretical value $L(r) = r$ for a stationary Poisson process

together with columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $L(r)$ obtained by the edge corrections named.

Variance approximations

If the argument `var.approx=TRUE` is given, the return value includes columns `rip` and `ls` containing approximations to the variance of $\hat{L}(r)$ under CSR. These are obtained by the delta method from the variance approximations described in `Kest`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Besag, J. (1977) Discussion of Dr Ripley's paper. *Journal of the Royal Statistical Society, Series B*, **39**, 193–195.

See Also

`Kest`, `pcf`

Examples

```
data(cells)
L <- Lest(cells)
plot(L, main="L function for cells")
```

`levelset` *Level Set of a Pixel Image*

Description

Given a pixel image, find all pixels which have values less than a specified threshold value (or greater than a threshold, etc), and assemble these pixels into a window.

Usage

```
levelset(X, thresh, compare="<=")
```

Arguments

`X`	A pixel image (object of class "im").
`thresh`	Threshold value. A single number or value compatible with the pixel values in `X`.
`compare`	Character string specifying one of the comparison operators `"<"`, `">"`, `"=="`, `"<="`, `">="`, `"!="`.

Details

If X is a pixel image with numeric values, then levelset(X, thresh) finds the region of space where the pixel values are less than or equal to the threshold value thresh. This region is returned as a spatial window.

The argument compare specifies how the pixel values should be compared with the threshold value. Instead of requiring pixel values to be less than or equal to thresh, you can specify that they must be less than (<), greater than (>), equal to (==), greater than or equal to (>=), or not equal to (!=) the threshold value thresh.

If X has non-numeric pixel values (for example, logical or factor values) it is advisable to use only the comparisons == and !=, unless you really know what you are doing.

For more complicated logical comparisons, see solutionset.

Value

A spatial window (object of class "owin", see owin.object) containing the pixels satisfying the constraint.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

im.object, as.owin, solutionset.

Examples

```
# test image
X <- as.im(function(x,y) { x^2 - y^2 }, unit.square())

W <- levelset(X, 0.2)
W <- levelset(X, -0.3, ">")

# compute area of level set
area(levelset(X, 0.1))
```

---

leverage.ppm *Leverage Measure for Spatial Point Process Model*

Description

Computes the leverage measure for a fitted spatial point process model.

Usage

```
leverage(model, ...)

## S3 method for class 'ppm'
leverage(model, ..., drop = FALSE, iScore=NULL, iHessian=NULL, iArgs=NULL)
```

Arguments

`model`	Fitted point process model (object of class "ppm").
`...`	Ignored, except for the arguments `dimyx` and `eps` which are passed to `as.mask` to control the spatial resolution of the result.
`drop`	Logical. Whether to include (`drop=FALSE`) or exclude (`drop=TRUE`) contributions from quadrature points that were not used to fit the model.
`iScore,iHessian`	Components of the score vector and Hessian matrix for the irregular parameters, if required. See Details.
`iArgs`	List of extra arguments for the functions `iScore`, `iHessian` if required.

Details

The function `leverage` is generic, and `leverage.ppm` is the method for objects of class "ppm".

Given a fitted spatial point process model `model`, the function `leverage.ppm` computes the leverage of the model, described in Baddeley, Chang and Song (2013).

The leverage of a spatial point process model is a function of spatial location, and is typically displayed as a colour pixel image. The leverage value $h(u)$ at a spatial location u represents the change in the fitted trend of the fitted point process model that would have occurred if a data point were to have occurred at the location u. A relatively large value of $h()$ indicates a part of the space where the data have a *potentially* strong effect on the fitted model (specifically, a strong effect on the intensity or trend of the fitted model) due to the values of the covariates.

If the point process model trend has irregular parameters that were fitted (using `ippm`) then the leverage calculation requires the first and second derivatives of the log trend with respect to the irregular parameters. The argument `iScore` should be a list, with one entry for each irregular parameter, of R functions that compute the partial derivatives of the log trend (i.e. log intensity or log conditional intensity) with respect to each irregular parameter. The argument `iHessian` should be a list, with p^2 entries where p is the number of irregular parameters, of R functions that compute the second order partial derivatives of the log trend with respect to each pair of irregular parameters.

The result of `leverage.ppm` is an object of class "leverage.ppm". It can be plotted (by `plot.leverage.ppm`) or converted to a pixel image by `as.im` (see `as.im.leverage.ppm`).

Value

An object of class "leverage.ppm" that can be plotted (by `plot.leverage.ppm`). There are also methods for `persp`, `print`, `[`, `as.im`, `as.function` and `as.owin`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Chang, Y.M. and Song, Y. (2013) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics* **40**, 86–104.

See Also

`influence.ppm`, `dfbetas.ppm`, `ppmInfluence`, `plot.leverage.ppm` `as.function.leverage.ppm`

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X ~x+y)
plot(leverage(fit))
```

`lgcp.estK` *Fit a Log-Gaussian Cox Point Process by Minimum Contrast*

Description

Fits a log-Gaussian Cox point process model to a point pattern dataset by the Method of Minimum Contrast.

Usage

```
lgcp.estK(X, startpar=c(var=1,scale=1),
             covmodel=list(model="exponential"),
             lambda=NULL,
             q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...)
```

Arguments

`X`	Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
`startpar`	Vector of starting values for the parameters of the log-Gaussian Cox process model.
`covmodel`	Specification of the covariance model for the log-Gaussian field. See Details.
`lambda`	Optional. An estimate of the intensity of the point process.
`q,p`	Optional. Exponents for the contrast criterion.
`rmin, rmax`	Optional. The interval of r values for the contrast criterion.
`...`	Optional arguments passed to `optim` to control the optimisation algorithm. See Details.

Details

This algorithm fits a log-Gaussian Cox point process (LGCP) model to a point pattern dataset by the Method of Minimum Contrast, using the K function of the point pattern.

The shape of the covariance of the LGCP must be specified: the default is the exponential covariance function, but other covariance models can be selected.

The argument `X` can be either

a point pattern: An object of class `"ppp"` representing a point pattern dataset. The K function of the point pattern will be computed using `Kest`, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class `"fv"` containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the K function, and this object should have been obtained by a call to `Kest` or one of its relatives.

The algorithm fits a log-Gaussian Cox point process (LGCP) model to X, by finding the parameters of the LGCP model which give the closest match between the theoretical K function of the LGCP model and the observed K function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The model fitted is a stationary, isotropic log-Gaussian Cox process (Møller and Waagepetersen, 2003, pp. 72-76). To define this process we start with a stationary Gaussian random field Z in the two-dimensional plane, with constant mean μ and covariance function $C(r)$. Given Z, we generate a Poisson point process Y with intensity function $\lambda(u) = \exp(Z(u))$ at location u. Then Y is a log-Gaussian Cox process.

The K-function of the LGCP is

$$K(r) = \int_0^r 2\pi s \exp(C(s))\, ds.$$

The intensity of the LGCP is

$$\lambda = \exp(\mu + \frac{C(0)}{2}).$$

The covariance function $C(r)$ is parametrised in the form

$$C(r) = \sigma^2 c(r/\alpha)$$

where σ^2 and α are parameters controlling the strength and the scale of autocorrelation, respectively, and $c(r)$ is a known covariance function determining the shape of the covariance. The strength and scale parameters σ^2 and α will be estimated by the algorithm as the values var and scale respectively. The template covariance function $c(r)$ must be specified as explained below.

In this algorithm, the Method of Minimum Contrast is first used to find optimal values of the parameters σ^2 and α. Then the remaining parameter μ is inferred from the estimated intensity λ.

The template covariance function $c(r)$ is specified using the argument covmodel. This should be of the form list(model="modelname", ...) where modelname is a string identifying the template model as explained below, and ... are optional arguments of the form tag=value giving the values of parameters controlling the *shape* of the template model. The default is the exponential covariance $c(r) = e^{-r}$ so that the scaled covariance is

$$C(r) = \sigma^2 e^{-r/\alpha}.$$

To determine the template model, the string "modelname" will be prefixed by "RM" and the code will search for a function of this name in the **RandomFields** package. For a list of available models see RMmodel in the **RandomFields** package. For example the Matern covariance with exponent $\nu = 0.3$ is specified by covmodel=list(model="matern", nu=0.3) corresponding to the function RMmatern in the **RandomFields** package.

If the argument lambda is provided, then this is used as the value of λ. Otherwise, if X is a point pattern, then λ will be estimated from X. If X is a summary statistic and lambda is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as NA.

The remaining arguments rmin,rmax,q,p control the method of minimum contrast; see mincontrast.

The optimisation algorithm can be controlled through the additional arguments "..." which are passed to the optimisation function optim. For example, to constrain the parameter values to a certain range, use the argument method="L-BFGS-B" to select an optimisation algorithm that respects box constraints, and use the arguments lower and upper to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following main components:

- `par` Vector of fitted parameter values.
- `fit` Function value table (object of class "fv") containing the observed values of the summary statistic (observed) and the theoretical values of the summary statistic computed from the fitted model parameters.

Note

This function is considerably slower than `lgcp.estpcf` because of the computation time required for the integral in the K-function.

Computation can be accelerated, at the cost of less accurate results, by setting `spatstat.options(fastK.lgcp=TRUE)`.

Author(s)

Rasmus Waagepetersen <rw@math.auc.dk>. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu

Further modifications by Rasmus Waagepetersen and Shen Guochun, and by Ege Rubak <rubak@math.aau.dk>.

References

Møller, J, Syversveen, A. and Waagepetersen, R. (1998) Log Gaussian Cox Processes. *Scandinavian Journal of Statistics* **25**, 451–482.

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

`lgcp.estpcf` for alternative method of fitting LGCP.

`matclust.estK`, `thomas.estK` for other models.

`mincontrast` for the generic minimum contrast fitting algorithm, including important parameters that affect the accuracy of the fit.

`RMmodel` in the **RandomFields** package, for covariance function models.

`Kest` for the K function.

Examples

```
if(interactive()) {
  u <- lgcp.estK(redwood)
} else {
  # slightly faster - better starting point
  u <- lgcp.estK(redwood, c(var=1, scale=0.1))
}
u
plot(u)


if(FALSE) {
```

```
    ## takes several minutes!
    lgcp.estK(redwood, covmodel=list(model="matern", nu=0.3))
  }
```

---

lgcp.estpcf *Fit a Log-Gaussian Cox Point Process by Minimum Contrast*

Description

Fits a log-Gaussian Cox point process model to a point pattern dataset by the Method of Minimum Contrast using the pair correlation function.

Usage

```
lgcp.estpcf(X,
            startpar=c(var=1,scale=1),
            covmodel=list(model="exponential"),
            lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ..., pcfargs=list())
```

Arguments

X	Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the log-Gaussian Cox process model.
covmodel	Specification of the covariance model for the log-Gaussian field. See Details.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.
pcfargs	Optional list containing arguments passed to pcf.ppp to control the smoothing in the estimation of the pair correlation function.

Details

This algorithm fits a log-Gaussian Cox point process (LGCP) model to a point pattern dataset by the Method of Minimum Contrast, using the estimated pair correlation function of the point pattern.

The shape of the covariance of the LGCP must be specified: the default is the exponential covariance function, but other covariance models can be selected.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The pair correlation function of the point pattern will be computed using pcf, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the pair correlation function, and this object should have been obtained by a call to pcf or one of its relatives.

The algorithm fits a log-Gaussian Cox point process (LGCP) model to `X`, by finding the parameters of the LGCP model which give the closest match between the theoretical pair correlation function of the LGCP model and the observed pair correlation function. For a more detailed explanation of the Method of Minimum Contrast, see `mincontrast`.

The model fitted is a stationary, isotropic log-Gaussian Cox process (Møller and Waagepetersen, 2003, pp. 72-76). To define this process we start with a stationary Gaussian random field Z in the two-dimensional plane, with constant mean μ and covariance function $C(r)$. Given Z, we generate a Poisson point process Y with intensity function $\lambda(u) = \exp(Z(u))$ at location u. Then Y is a log-Gaussian Cox process.

The theoretical pair correlation function of the LGCP is

$$g(r) = \exp(C(s))$$

The intensity of the LGCP is

$$\lambda = \exp(\mu + \frac{C(0)}{2}).$$

The covariance function $C(r)$ takes the form

$$C(r) = \sigma^2 c(r/\alpha)$$

where σ^2 and α are parameters controlling the strength and the scale of autocorrelation, respectively, and $c(r)$ is a known covariance function determining the shape of the covariance. The strength and scale parameters σ^2 and α will be estimated by the algorithm. The template covariance function $c(r)$ must be specified as explained below.

In this algorithm, the Method of Minimum Contrast is first used to find optimal values of the parameters σ^2 and α. Then the remaining parameter μ is inferred from the estimated intensity λ.

The template covariance function $c(r)$ is specified using the argument `covmodel`. This should be of the form `list(model="modelname", ...)` where `modelname` is a string identifying the template model as explained below, and `...` are optional arguments of the form `tag=value` giving the values of parameters controlling the *shape* of the template model. The default is the exponential covariance $c(r) = e^{-r}$ so that the scaled covariance is

$$C(r) = \sigma^2 e^{-r/\alpha}.$$

To determine the template model, the string `"modelname"` will be prefixed by `"RM"` and the code will search for a function of this name in the **RandomFields** package. For a list of available models see `RMmodel` in the **RandomFields** package. For example the Matern covariance with exponent $\nu = 0.3$ is specified by `covmodel=list(model="matern", nu=0.3)` corresponding to the function `RMmatern` in the **RandomFields** package.

If the argument `lambda` is provided, then this is used as the value of λ. Otherwise, if `X` is a point pattern, then λ will be estimated from `X`. If `X` is a summary statistic and `lambda` is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as `NA`.

The remaining arguments `rmin,rmax,q,p` control the method of minimum contrast; see `mincontrast`.

The optimisation algorithm can be controlled through the additional arguments `"..."` which are passed to the optimisation function `optim`. For example, to constrain the parameter values to a certain range, use the argument `method="L-BFGS-B"` to select an optimisation algorithm that respects box constraints, and use the arguments `lower` and `upper` to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following main components:

par	Vector of fitted parameter values.
fit	Function value table (object of class "fv") containing the observed values of the summary statistic (observed) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> with modifications by Shen Guochun and Rasmus Waagepetersen <rw@math.auc.dk> and Ege Rubak <rubak@math.aau.dk>.

References

Møller, J., Syversveen, A. and Waagepetersen, R. (1998) Log Gaussian Cox Processes. *Scandinavian Journal of Statistics* **25**, 451–482.

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

lgcp.estK for alternative method of fitting LGCP.

matclust.estpcf, thomas.estpcf for other models.

mincontrast for the generic minimum contrast fitting algorithm, including important parameters that affect the accuracy of the fit.

RMmodel in the **RandomFields** package, for covariance function models.

pcf for the pair correlation function.

Examples

```
data(redwood)
u <- lgcp.estpcf(redwood, c(var=1, scale=0.1))
u
plot(u)
if(require(RandomFields)) {
  lgcp.estpcf(redwood, covmodel=list(model="matern", nu=0.3))
}
```

lineardirichlet *Dirichlet Tessellation on a Linear Network*

Description

Given a point pattern on a linear network, compute the Dirichlet (or Voronoi or Thiessen) tessellation induced by the points.

Usage

```
lineardirichlet(X)
```

Arguments

X	Point pattern on a linear network (object of class "lpp").

Details

The Dirichlet tessellation induced by a point pattern `X` on a linear network `L` is a partition of `L` into subsets. The subset `L[i]` associated with the data point `X[i]` is the part of `L` lying closer to `X[i]` than to any other data point `X[j]`, where distance is measured by the shortest path.

Value

A tessellation on a linear network (object of class "lintess").

Missing tiles

If the linear network is not connected, and if one of the connected components contains no data points, then the Dirichlet tessellation is mathematically undefined inside this component. The resulting tessellation object includes a tile with label `NA`, which contains this component of the network. A plot of the tessellation will not show this tile.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

lintess

Examples

```
X <- runiflpp(5, simplenet)
plot(lineardirichlet(X), lwd=3)
points(X)
```

lineardisc *Compute Disc of Given Radius in Linear Network*

Description

Computes the 'disc' of given radius and centre in a linear network.

Usage

```
lineardisc(L, x = locator(1), r, plotit = TRUE,
           cols=c("blue", "red","green"))

countends(L, x = locator(1), r, toler=NULL)
```

Arguments

L	Linear network (object of class "linnet").
x	Location of centre of disc. Either a point pattern (object of class "ppp") containing exactly 1 point, or a numeric vector of length 2.
r	Radius of disc.
plotit	Logical. Whether to plot the disc.
cols	Colours for plotting the disc. A numeric or character vector of length 3 specifying the colours of the disc centre, disc lines and disc endpoints respectively.
toler	Optional. Distance threshold for countends. See Details. There is a sensible default.

Details

The 'disc' $B(u, r)$ of centre x and radius r in a linear network L is the set of all points u in L such that the shortest path distance from x to u is less than or equal to r. This is a union of line segments contained in L.

The *relative boundary* of the disc $B(u, r)$ is the set of points v such that the shortest path distance from x to u is *equal* to r.

The function lineardisc computes the disc of radius r and its relative boundary, optionally plots them, and returns them. The faster function countends simply counts the number of points in the relative boundary.

The optional threshold toler is used to suppress numerical errors in countends. If the distance from u to a network vertex v is between r-toler and r+toler, the vertex will be treated as lying on the relative boundary.

Value

The value of lineardisc is a list with two entries:

lines	Line segment pattern (object of class "psp") representing the interior disc
endpoints	Point pattern (object of class "ppp") representing the relative boundary of the disc.

The value of countends is an integer giving the number of points in the relative boundary.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Ang, Q.W. (2010) *Statistical methodology for events on a network.* Master's thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

See Also

linnet

Examples

```
 # letter 'A'
 v <- ppp(x=(-2):2, y=3*c(0,1,2,1,0), c(-3,3), c(-1,7))
 edg <- cbind(1:4, 2:5)
 edg <- rbind(edg, c(2,4))
 letterA <- linnet(v, edges=edg)

lineardisc(letterA, c(0,3), 1.6)
# count the endpoints
countends(letterA, c(0,3), 1.6)
# cross-check (slower)
en <- lineardisc(letterA, c(0,3), 1.6, plotit=FALSE)$endpoints
npoints(en)
```

---

linearK *Linear K Function*

Description

Computes an estimate of the linear K function for a point pattern on a linear network.

Usage

```
linearK(X, r=NULL, ..., correction="Ang", ratio=FALSE)
```

Arguments

X	Point pattern on linear network (object of class "lpp").
r	Optional. Numeric vector of values of the function argument r. There is a sensible default.
...	Ignored.
correction	Geometry correction. Either "none" or "Ang". See Details.
ratio	Logical. If TRUE, the numerator and denominator of the estimate will also be saved, for use in analysing replicated point patterns.

Details

This command computes the linear K function from point pattern data on a linear network.

If correction="none", the calculations do not include any correction for the geometry of the linear network. The result is the network K function as defined by Okabe and Yamada (2001).

If correction="Ang", the pair counts are weighted using Ang's correction (Ang, 2010; Ang et al, 2012).

Value

Function value table (object of class "fv").

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Ang, Q.W. (2010) Statistical methodology for spatial point patterns on a linear network. MSc thesis, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

Okabe, A. and Yamada, I. (2001) The K-function method on a network and its computational implementation. *Geographical Analysis* **33**, 271-290.

See Also

compileK, lpp

Examples

```
data(simplenet)
X <- rpoislpp(5, simplenet)
linearK(X)
linearK(X, correction="none")
```

---

linearKcross — *Multitype K Function (Cross-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the multitype K function which counts the expected number of points of type j within a given distance of a point of type i.

Usage

```
linearKcross(X, i, j, r=NULL, ..., correction="Ang")
```

Arguments

X	The observed point pattern, from which an estimate of the cross type K function $K_{ij}(r)$ will be computed. An object of class "lpp" which must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	Number or character string identifying the type (mark value) of the points in X from which distances are measured. Defaults to the first level of marks(X).
j	Number or character string identifying the type (mark value) of the points in X to which distances are measured. Defaults to the second level of marks(X).
r	numeric vector. The values of the argument r at which the K-function $K_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
correction	Geometry correction. Either "none" or "Ang". See Details.
...	Ignored.

Details

This is a counterpart of the function Kcross for a point pattern on a linear network (object of class "lpp").

The arguments i and j will be interpreted as levels of the factor marks(X). If i and j are missing, they default to the first and second level of the marks factor, respectively.

The argument r is the vector of values for the distance r at which $K_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class "fv" (see fv.object).

Warnings

The arguments i and j are interpreted as levels of the factor marks(X). Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (to appear) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), In press.

See Also

linearKdot, linearK.

Examples

```
data(chicago)
K <- linearKcross(chicago, "assault", "robbery")
```

linearKcross.inhom — *Inhomogeneous multitype K Function (Cross-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the inhomogeneous multitype K function which counts the expected number of points of type j within a given distance of a point of type i.

Usage

```
linearKcross.inhom(X, i, j, lambdaI, lambdaJ,
                   r=NULL, ..., correction="Ang", normalise=TRUE)
```

Arguments

`X`	The observed point pattern, from which an estimate of the cross type K function $K_{ij}(r)$ will be computed. An object of class `"lpp"` which must be a multitype point pattern (a marked point pattern whose marks are a factor).
`i`	Number or character string identifying the type (mark value) of the points in `X` from which distances are measured. Defaults to the first level of `marks(X)`.
`j`	Number or character string identifying the type (mark value) of the points in `X` to which distances are measured. Defaults to the second level of `marks(X)`.
`lambdaI`	Intensity values for the points of type `i`. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).
`lambdaJ`	Intensity values for the points of type `j`. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).
`r`	numeric vector. The values of the argument r at which the K-function $K_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`...`	Arguments passed to `lambdaI` and `lambdaJ` if they are functions.
`normalise`	Logical. If `TRUE` (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the points of type `i`), which reduces the sampling variability. If `FALSE`, the denominator is the length of the network.

Details

This is a counterpart of the function `Kcross.inhom` for a point pattern on a linear network (object of class `"lpp"`).

The arguments `i` and `j` will be interpreted as levels of the factor `marks(X)`. If `i` and `j` are missing, they default to the first and second level of the marks factor, respectively.

The argument `r` is the vector of values for the distance r at which $K_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

If `lambdaI` or `lambdaJ` is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting `update=FALSE`.

Value

An object of class `"fv"` (see `fv.object`).

Warnings

The arguments `i` and `j` are interpreted as levels of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (to appear) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), In press.

See Also

`linearKdot`, `linearK`.

Examples

```
   lam <- table(marks(chicago))/(summary(chicago)$totlength)
   lamI <- function(x,y,const=lam[["assault"]]){ rep(const, length(x)) }
   lamJ <- function(x,y,const=lam[["robbery"]]){ rep(const, length(x)) }

   K <- linearKcross.inhom(chicago, "assault", "robbery", lamI, lamJ)

   ## Not run:
     fit <- lppm(chicago, ~marks + x)
     linearKcross.inhom(chicago, "assault", "robbery", fit, fit)

## End(Not run)
```

`linearKdot` *Multitype K Function (Dot-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the multitype K function which counts the expected number of points (of any type) within a given distance of a point of type i.

Usage

```
linearKdot(X, i, r=NULL, ..., correction="Ang")
```

Arguments

X	The observed point pattern, from which an estimate of the dot type K function $K_{i\bullet}(r)$ will be computed. An object of class "lpp" which must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	Number or character string identifying the type (mark value) of the points in X from which distances are measured. Defaults to the first level of marks(X).
r	numeric vector. The values of the argument r at which the K-function $K_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
correction	Geometry correction. Either "none" or "Ang". See Details.
...	Ignored.

Details

This is a counterpart of the function Kdot for a point pattern on a linear network (object of class "lpp").

The argument i will be interpreted as levels of the factor marks(X). If i is missing, it defaults to the first level of the marks factor.

The argument r is the vector of values for the distance r at which $K_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class "fv" (see fv.object).

Warnings

The argument i is interpreted as a level of the factor marks(X). Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (to appear) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), In press.

See Also

Kdot, linearKcross, linearK.

Examples

```
data(chicago)
K <- linearKdot(chicago, "assault")
```

linearKdot.inhom	*Inhomogeneous multitype K Function (Dot-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the inhomogeneous multitype K function which counts the expected number of points (of any type) within a given distance of a point of type i.

Usage

```
linearKdot.inhom(X, i, lambdaI, lambdadot, r=NULL, ...,
                 correction="Ang", normalise=TRUE)
```

Arguments

X	The observed point pattern, from which an estimate of the dot type K function $K_{i\bullet}(r)$ will be computed. An object of class "lpp" which must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	Number or character string identifying the type (mark value) of the points in X from which distances are measured. Defaults to the first level of marks(X).
lambdaI	Intensity values for the points of type i. Either a numeric vector, a function, a pixel image (object of class "im" or "linim") or a fitted point process model (object of class "ppm" or "lppm").
lambdadot	Intensity values for all points of X. Either a numeric vector, a function, a pixel image (object of class "im" or "linim") or a fitted point process model (object of class "ppm" or "lppm").
r	numeric vector. The values of the argument r at which the K-function $K_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
correction	Geometry correction. Either "none" or "Ang". See Details.
...	Arguments passed to lambdaI and lambdadot if they are functions.
normalise	Logical. If TRUE (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the points of type i), which reduces the sampling variability. If FALSE, the denominator is the length of the network.

Details

This is a counterpart of the function Kdot.inhom for a point pattern on a linear network (object of class "lpp").

The argument i will be interpreted as levels of the factor marks(X). If i is missing, it defaults to the first level of the marks factor.

The argument r is the vector of values for the distance r at which $K_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

If lambdaI or lambdadot is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting update=FALSE.

Value

An object of class "fv" (see fv.object).

Warnings

The argument i is interpreted as a level of the factor marks(X). Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (to appear) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), In press.

See Also

linearKdot, linearK.

Examples

```
   lam <- table(marks(chicago))/(summary(chicago)$totlength)
   lamI <- function(x,y,const=lam[["assault"]]){ rep(const, length(x)) }
   lam. <- function(x,y,const=sum(lam)){ rep(const, length(x)) }

   K <- linearKdot.inhom(chicago, "assault", lamI, lam.)

   ## Not run:
     fit <- lppm(chicago, ~marks + x)
     linearKdot.inhom(chicago, "assault", fit, fit)

## End(Not run)
```

---

linearKinhom *Inhomogeneous Linear K Function*

Description

Computes an estimate of the inhomogeneous linear K function for a point pattern on a linear network.

Usage

```
linearKinhom(X, lambda=NULL, r=NULL, ..., correction="Ang",
               normalise=TRUE, normpower=1,
      update=TRUE, leaveoneout=TRUE, ratio=FALSE)
```

Arguments

- `X` — Point pattern on linear network (object of class `"lpp"`).
- `lambda` — Intensity values for the point pattern. Either a numeric vector, a `function`, a pixel image (object of class `"im"` or `"linim"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).
- `r` — Optional. Numeric vector of values of the function argument r. There is a sensible default.
- `...` — Ignored.
- `correction` — Geometry correction. Either `"none"` or `"Ang"`. See Details.
- `normalise` — Logical. If `TRUE` (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the data points, raised to `normpower`), which reduces the sampling variability. If `FALSE`, the denominator is the length of the network.
- `normpower` — Integer (usually either 1 or 2). Normalisation power. See Details.
- `update` — Logical value indicating what to do when `lambda` is a fitted model (class `"lppm"` or `"ppm"`). If `update=TRUE` (the default), the model will first be refitted to the data `X` (using `update.lppm` or `update.ppm`) before the fitted intensity is computed. If `update=FALSE`, the fitted intensity of the model will be computed without re-fitting it to `X`.
- `leaveoneout` — Logical value (passed to `fitted.lppm` or `fitted.ppm`) specifying whether to use a leave-one-out rule when calculating the intensity, when `lambda` is a fitted model. Supported only when `update=TRUE`.
- `ratio` — Logical. If `TRUE`, the numerator and denominator of the estimate will also be saved, for use in analysing replicated point patterns.

Details

This command computes the inhomogeneous version of the linear K function from point pattern data on a linear network.

If `lambda = NULL` the result is equivalent to the homogeneous K function `linearK`. If `lambda` is given, then it is expected to provide estimated values of the intensity of the point process at each point of `X`. The argument `lambda` may be a numeric vector (of length equal to the number of points in `X`), or a `function(x,y)` that will be evaluated at the points of `X` to yield numeric values, or a pixel image (object of class `"im"`) or a fitted point process model (object of class `"ppm"` or `"lppm"`).

If `lambda` is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting `update=FALSE`.

If `correction="none"`, the calculations do not include any correction for the geometry of the linear network. If `correction="Ang"`, the pair counts are weighted using Ang's correction (Ang, 2010).

Each estimate is initially computed as

$$\widehat{K}_{\text{inhom}}(r) = \frac{1}{\text{length}(L)} \sum_i \sum_j \frac{1\{d_{ij} \le r\} e(x_i, x_j)}{\lambda(x_i)\lambda(x_j)}$$

where `L` is the linear network, d_{ij} is the distance between points x_i and x_j, and $e(x_i, x_j)$ is a weight. If `correction="none"` then this weight is equal to 1, while if `correction="Ang"` the weight is $e(x_i, x_j, r) = 1/m(x_i, d_{ij})$ where $m(u, t)$ is the number of locations on the network that lie exactly t units distant from location u by the shortest path.

If normalise=TRUE (the default), then the estimates described above are multiplied by $c^{\mathrm{normpower}}$ where $c = \mathrm{length}(L)/\sum(1/\lambda(x_i))$. This rescaling reduces the variability and bias of the estimate in small samples and in cases of very strong inhomogeneity. The default value of normpower is 1 (for consistency with previous versions of **spatstat**) but the most sensible value is 2, which would correspond to rescaling the lambda values so that $\sum(1/\lambda(x_i)) = \mathrm{area}(W)$.

Value

Function value table (object of class "fv").

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Ang, Q.W. (2010) Statistical methodology for spatial point patterns on a linear network. MSc thesis, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

See Also

lpp

Examples

```
data(simplenet)
X <- rpoislpp(5, simplenet)
fit <- lppm(X ~x)
K <- linearKinhom(X, lambda=fit)
plot(K)
```

---

linearmarkconnect *Mark Connection Function for Multitype Point Pattern on Linear Network*

Description

For a multitype point pattern on a linear network, estimate the mark connection function from points of type i to points of type j.

Usage

```
linearmarkconnect(X, i, j, r=NULL, ...)
```

Arguments

- `X` The observed point pattern, from which an estimate of the mark connection function $p_{ij}(r)$ will be computed. An object of class "lpp" which must be a multitype point pattern (a marked point pattern whose marks are a factor).
- `i` Number or character string identifying the type (mark value) of the points in `X` from which distances are measured. Defaults to the first level of `marks(X)`.
- `j` Number or character string identifying the type (mark value) of the points in `X` to which distances are measured. Defaults to the second level of `marks(X)`.
- `r` numeric vector. The values of the argument r at which the function $p_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
- `...` Arguments passed to `linearpcfcross` and `linearpcf`.

Details

This is a counterpart of the function `markconnect` for a point pattern on a linear network (object of class "lpp").

The argument `i` will be interpreted as levels of the factor `marks(X)`. If `i` is missing, it defaults to the first level of the marks factor.

The argument `r` is the vector of values for the distance r at which $p_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class "fv" (see `fv.object`).

Warnings

The argument `i` is interpreted as a level of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (to appear) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), In press.

See Also

`linearpcfcross`, `linearpcf`, `linearmarkequal`, `markconnect`.

Examples

```
   pab <- linearmarkconnect(chicago, "assault", "burglary")
  ## Not run:
   plot(alltypes(chicago, linearmarkconnect))

## End(Not run)
```

`linearmarkequal` *Mark Connection Function for Multitype Point Pattern on Linear Network*

Description

For a multitype point pattern on a linear network, estimate the mark connection function from points of type i to points of type j.

Usage

```
linearmarkequal(X, r=NULL, ...)
```

Arguments

X	The observed point pattern, from which an estimate of the mark connection function $p_{ij}(r)$ will be computed. An object of class "lpp" which must be a multitype point pattern (a marked point pattern whose marks are a factor).
r	numeric vector. The values of the argument r at which the function $p_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
...	Arguments passed to `linearpcfcross` and `linearpcf`.

Details

This is the mark equality function for a point pattern on a linear network (object of class "lpp").

The argument `r` is the vector of values for the distance r at which $p_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class "fv" (see `fv.object`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (to appear) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), In press.

See Also

`linearpcfcross`, `linearpcf`, `linearmarkconnect`, `markconnect`.

Examples

```
if(interactive()) {
  X <- chicago
} else {
  X <- runiflpp(20, simplenet) %mark% sample(c("A","B"), 20,
  replace=TRUE)
}
p <- linearmarkequal(X)
```

---

linearpcf — *Linear Pair Correlation Function*

Description

Computes an estimate of the linear pair correlation function for a point pattern on a linear network.

Usage

```
linearpcf(X, r=NULL, ..., correction="Ang", ratio=FALSE)
```

Arguments

`X`	Point pattern on linear network (object of class "lpp").
`r`	Optional. Numeric vector of values of the function argument r. There is a sensible default.
`...`	Arguments passed to `density.default` to control the smoothing.
`correction`	Geometry correction. Either "none" or "Ang". See Details.
`ratio`	Logical. If `TRUE`, the numerator and denominator of each estimate will also be saved, for use in analysing replicated point patterns.

Details

This command computes the linear pair correlation function from point pattern data on a linear network.

The pair correlation function is estimated from the shortest-path distances between each pair of data points, using the fixed-bandwidth kernel smoother `density.default`, with a bias correction at each end of the interval of r values. To switch off the bias correction, set `endcorrect=FALSE`.

The bandwidth for smoothing the pairwise distances is determined by arguments `...` passed to `density.default`, mainly the arguments `bw` and `adjust`. The default is to choose the bandwidth by Silverman's rule of thumb `bw="nrd0"` explained in `density.default`.

If `correction="none"`, the calculations do not include any correction for the geometry of the linear network. The result is an estimate of the first derivative of the network K function defined by Okabe and Yamada (2001).

If `correction="Ang"`, the pair counts are weighted using Ang's correction (Ang, 2010). The result is an estimate of the pair correlation function in the linear network.

Value

Function value table (object of class "fv").

If ratio=TRUE then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $g(r)$.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Ang, Q.W. (2010) Statistical methodology for spatial point patterns on a linear network. MSc thesis, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

Okabe, A. and Yamada, I. (2001) The K-function method on a network and its computational implementation. *Geographical Analysis* **33**, 271-290.

See Also

linearK, linearpcfinhom, lpp

Examples

```
data(simplenet)
X <- rpoislpp(5, simplenet)
linearpcf(X)
linearpcf(X, correction="none")
```

---

linearpcfcross   *Multitype Pair Correlation Function (Cross-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the multitype pair correlation function from points of type i to points of type j.

Usage

```
linearpcfcross(X, i, j, r=NULL, ..., correction="Ang")
```

Arguments

X	The observed point pattern, from which an estimate of the i-to-any pair correlation function $g_{ij}(r)$ will be computed. An object of class "lpp" which must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	Number or character string identifying the type (mark value) of the points in X from which distances are measured. Defaults to the first level of marks(X).

`j`	Number or character string identifying the type (mark value) of the points in `X` to which distances are measured. Defaults to the second level of `marks(X)`.
`r`	numeric vector. The values of the argument r at which the function $g_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`correction`	Geometry correction. Either `"none"` or `"Ang"`. See Details.
`...`	Arguments passed to `density.default` to control the kernel smoothing.

Details

This is a counterpart of the function `pcfcross` for a point pattern on a linear network (object of class `"lpp"`).

The argument `i` will be interpreted as levels of the factor `marks(X)`. If `i` is missing, it defaults to the first level of the marks factor.

The argument `r` is the vector of values for the distance r at which $g_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class `"fv"` (see `fv.object`).

Warnings

The argument `i` is interpreted as a level of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (to appear) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), In press.

See Also

`linearpcfdot`, `linearpcf`, `pcfcross`.

Examples

```
data(chicago)
g <- linearpcfcross(chicago, "assault")
```

linearpcfcross.inhom *Inhomogeneous Multitype Pair Correlation Function (Cross-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the inhomogeneous multitype pair correlation function from points of type i to points of type j.

Usage

```
linearpcfcross.inhom(X, i, j, lambdaI, lambdaJ, r=NULL, ...,
                     correction="Ang", normalise=TRUE)
```

Arguments

X	The observed point pattern, from which an estimate of the i-to-any pair correlation function $g_{ij}(r)$ will be computed. An object of class "lpp" which must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	Number or character string identifying the type (mark value) of the points in X from which distances are measured. Defaults to the first level of marks(X).
j	Number or character string identifying the type (mark value) of the points in X to which distances are measured. Defaults to the second level of marks(X).
lambdaI	Intensity values for the points of type i. Either a numeric vector, a function, a pixel image (object of class "im" or "linim") or a fitted point process model (object of class "ppm" or "lppm").
lambdaJ	Intensity values for the points of type j. Either a numeric vector, a function, a pixel image (object of class "im" or "linim") or a fitted point process model (object of class "ppm" or "lppm").
r	numeric vector. The values of the argument r at which the function $g_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
correction	Geometry correction. Either "none" or "Ang". See Details.
...	Arguments passed to density.default to control the kernel smoothing.
normalise	Logical. If TRUE (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the points of type i), which reduces the sampling variability. If FALSE, the denominator is the length of the network.

Details

This is a counterpart of the function pcfcross.inhom for a point pattern on a linear network (object of class "lpp").

The argument i will be interpreted as levels of the factor marks(X). If i is missing, it defaults to the first level of the marks factor.

The argument r is the vector of values for the distance r at which $g_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

If `lambdaI` or `lambdaJ` is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting `update=FALSE`.

Value

An object of class `"fv"` (see `fv.object`).

Warnings

The argument `i` is interpreted as a level of the factor `marks(X)`. Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (to appear) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), In press.

See Also

`linearpcfdot`, `linearpcf`, `pcfcross.inhom`.

Examples

```
   lam <- table(marks(chicago))/(summary(chicago)$totlength)
   lamI <- function(x,y,const=lam[["assault"]]){ rep(const, length(x)) }
   lamJ <- function(x,y,const=lam[["robbery"]]){ rep(const, length(x)) }

   g <- linearpcfcross.inhom(chicago, "assault", "robbery", lamI, lamJ)

   ## Not run:
     fit <- lppm(chicago, ~marks + x)
     linearpcfcross.inhom(chicago, "assault", "robbery", fit, fit)

## End(Not run)
```

`linearpcfdot` *Multitype Pair Correlation Function (Dot-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the multitype pair correlation function from points of type i to points of any type.

Usage

```
linearpcfdot(X, i, r=NULL, ..., correction="Ang")
```

Arguments

X — The observed point pattern, from which an estimate of the i-to-any pair correlation function $g_{i\bullet}(r)$ will be computed. An object of class "lpp" which must be a multitype point pattern (a marked point pattern whose marks are a factor).

i — Number or character string identifying the type (mark value) of the points in X from which distances are measured. Defaults to the first level of marks(X).

r — numeric vector. The values of the argument r at which the function $g_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.

correction — Geometry correction. Either "none" or "Ang". See Details.

... — Arguments passed to density.default to control the kernel smoothing.

Details

This is a counterpart of the function pcfdot for a point pattern on a linear network (object of class "lpp").

The argument i will be interpreted as levels of the factor marks(X). If i is missing, it defaults to the first level of the marks factor.

The argument r is the vector of values for the distance r at which $g_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class "fv" (see fv.object).

Warnings

The argument i is interpreted as a level of the factor marks(X). Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (to appear) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), In press.

See Also

linearpcfcross, linearpcf. pcfcross.

Examples

```
data(chicago)
g <- linearpcfdot(chicago, "assault")
```

linearpcfdot.inhom   *Inhomogeneous Multitype Pair Correlation Function (Dot-type) for Linear Point Pattern*

Description

For a multitype point pattern on a linear network, estimate the inhomogeneous multitype pair correlation function from points of type i to points of any type.

Usage

```
linearpcfdot.inhom(X, i, lambdaI, lambdadot, r=NULL, ...,
                   correction="Ang", normalise=TRUE)
```

Arguments

X	The observed point pattern, from which an estimate of the i-to-any pair correlation function $g_{i\bullet}(r)$ will be computed. An object of class "lpp" which must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	Number or character string identifying the type (mark value) of the points in X from which distances are measured. Defaults to the first level of marks(X).
lambdaI	Intensity values for the points of type i. Either a numeric vector, a function, a pixel image (object of class "im" or "linim") or a fitted point process model (object of class "ppm" or "lppm").
lambdadot	Intensity values for all points of X. Either a numeric vector, a function, a pixel image (object of class "im" or "linim") or a fitted point process model (object of class "ppm" or "lppm").
r	numeric vector. The values of the argument r at which the function $g_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
correction	Geometry correction. Either "none" or "Ang". See Details.
...	Arguments passed to density.default to control the kernel smoothing.
normalise	Logical. If TRUE (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the points of type i), which reduces the sampling variability. If FALSE, the denominator is the length of the network.

Details

This is a counterpart of the function pcfdot.inhom for a point pattern on a linear network (object of class "lpp").

The argument i will be interpreted as levels of the factor marks(X). If i is missing, it defaults to the first level of the marks factor.

The argument r is the vector of values for the distance r at which $g_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

If lambdaI or lambdadot is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting update=FALSE.

Value

An object of class "fv" (see fv.object).

Warnings

The argument i is interpreted as a level of the factor marks(X). Beware of the usual trap with factors: numerical values are not interpreted in the same way as character values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Baddeley, A, Jammalamadaka, A. and Nair, G. (to appear) Multitype point process analysis of spines on the dendrite network of a neuron. *Applied Statistics* (Journal of the Royal Statistical Society, Series C), In press.

See Also

linearpcfcross.inhom, linearpcfcross, pcfcross.inhom.

Examples

```
    lam <- table(marks(chicago))/(summary(chicago)$totlength)
    lamI <- function(x,y,const=lam[["assault"]]){ rep(const, length(x)) }
    lam. <- function(x,y,const=sum(lam)){ rep(const, length(x)) }

    g <- linearpcfdot.inhom(chicago, "assault", lamI, lam.)

    ## Not run:
      fit <- lppm(chicago, ~marks + x)
      linearpcfdot.inhom(chicago, "assault", fit, fit)

## End(Not run)
```

---

linearpcfinhom *Inhomogeneous Linear Pair Correlation Function*

Description

Computes an estimate of the inhomogeneous linear pair correlation function for a point pattern on a linear network.

Usage

```
linearpcfinhom(X, lambda=NULL, r=NULL, ..., correction="Ang",
                 normalise=TRUE, normpower=1,
        update = TRUE, leaveoneout = TRUE,
        ratio = FALSE)
```

Arguments

- `X` Point pattern on linear network (object of class "lpp").
- `lambda` Intensity values for the point pattern. Either a numeric vector, a `function`, a pixel image (object of class "im") or a fitted point process model (object of class "ppm" or "lppm").
- `r` Optional. Numeric vector of values of the function argument r. There is a sensible default.
- `...` Arguments passed to `density.default` to control the smoothing.
- `correction` Geometry correction. Either "none" or "Ang". See Details.
- `normalise` Logical. If `TRUE` (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the data points, raised to `normpower`), which reduces the sampling variability. If `FALSE`, the denominator is the length of the network.
- `normpower` Integer (usually either 1 or 2). Normalisation power. See explanation in `linearKinhom`.
- `update` Logical value indicating what to do when `lambda` is a fitted model (class "lppm" or "ppm"). If `update=TRUE` (the default), the model will first be refitted to the data `X` (using `update.lppm` or `update.ppm`) before the fitted intensity is computed. If `update=FALSE`, the fitted intensity of the model will be computed without re-fitting it to `X`.
- `leaveoneout` Logical value (passed to `fitted.lppm` or `fitted.ppm`) specifying whether to use a leave-one-out rule when calculating the intensity, when `lambda` is a fitted model. Supported only when `update=TRUE`.
- `ratio` Logical. If `TRUE`, the numerator and denominator of each estimate will also be saved, for use in analysing replicated point patterns.

Details

This command computes the inhomogeneous version of the linear pair correlation function from point pattern data on a linear network.

If `lambda = NULL` the result is equivalent to the homogeneous pair correlation function `linearpcf`. If `lambda` is given, then it is expected to provide estimated values of the intensity of the point process at each point of `X`. The argument `lambda` may be a numeric vector (of length equal to the number of points in `X`), or a `function(x,y)` that will be evaluated at the points of `X` to yield numeric values, or a pixel image (object of class "im") or a fitted point process model (object of class "ppm" or "lppm").

If `lambda` is a fitted point process model, the default behaviour is to update the model by re-fitting it to the data, before computing the fitted intensity. This can be disabled by setting `update=FALSE`.

If `correction="none"`, the calculations do not include any correction for the geometry of the linear network. If `correction="Ang"`, the pair counts are weighted using Ang's correction (Ang, 2010).

The bandwidth for smoothing the pairwise distances is determined by arguments `...` passed to `density.default`, mainly the arguments `bw` and `adjust`. The default is to choose the bandwidth by Silverman's rule of thumb `bw="nrd0"` explained in `density.default`.

Value

Function value table (object of class "fv").

If `ratio=TRUE` then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $g(r)$.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Ang, Q.W. (2010) Statistical methodology for spatial point patterns on a linear network. MSc thesis, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

Okabe, A. and Yamada, I. (2001) The K-function method on a network and its computational implementation. *Geographical Analysis* **33**, 271-290.

See Also

linearpcf, linearKinhom, lpp

Examples

```
data(simplenet)
X <- rpoislpp(5, simplenet)
fit <- lppm(X ~x)
K <- linearpcfinhom(X, lambda=fit)
plot(K)
```

---

linequad *Quadrature Scheme on a Linear Network*

Description

Generates a quadrature scheme (an object of class "quad") on a linear network.

Usage

```
linequad(X, Y, ..., eps = NULL, nd = 1000, random = FALSE)
```

Arguments

X	Data points. An object of class "lpp" or "ppp".
Y	Line segments on which the points of X lie. An object of class "psp". Required only when X is a "ppp" object.
...	Ignored.
eps	Optional. Spacing between successive dummy points along each segment.
nd	Optional. Total number of dummy points to be generated.
random	Logical value indicating whether the sequence of dummy points should start at a randomly-chosen position along each segment.

Details

This command generates a quadrature scheme (object of class "quad") from a pattern of points on a linear network.

Normally the user does not need to call `linequad` explicitly. It is invoked by **spatstat** functions when needed. A quadrature scheme is required by `lppm` in order to fit point process models to point pattern data on a linear network. A quadrature scheme is also used by `rhohat.lpp` and other functions.

In order to create the quadrature scheme, dummy points are placed along each line segment of the network. The dummy points are evenly-spaced with spacing `eps`. The default is `eps = totlen/nd` where `totlen` is the total length of all line segments in the network.

Value

A quadrature scheme (object of class "quad").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Greg McSwiggan and Suman Rakshit.

See Also

`lppm`

`linfun` *Function on a Linear Network*

Description

Create a function on a linear network.

Usage

```
linfun(f, L)
```

Arguments

f	A `function` in the R language.
L	A linear network (object of class "linnet") on which `f` is defined.

Details

This creates an object of class "linfun". This is a simple mechanism for handling a function defined on a linear network, to make it easier to display and manipulate.

`f` should be a `function` in the R language, with formal arguments `f(x,y,seg,tp)` or `f(x,y,seg,tp, ...)` where `x,y` are Cartesian coordinates of locations on the linear network, `seg, tp` are the local coordinates, and `...` are optional additional arguments.

The function `f` should be vectorised: that is, if `x,y,seg,tp` are numeric vectors of the same length `n`, then `v <- f(x,y,seg,tp)` should be a vector of length `n`.

`L` should be a linear network (object of class "linnet") inside which the function `f` is well-defined.

The result is a function g in the R language which belongs to the special class "linfun". This function can be called as g(X) where X is an "lpp" object, or called as g(x,y) or g(x,y,seg,tp) where x,y,seg,tp are coordinates. There are several methods for this class including print, plot and as.linim.

Value

A function in the R\ language. It also belongs to the class "linfun" which has methods for plot, print etc.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

methods.linfun for methods applicable to "linfun" objects.

distfun.lpp, nnfun.lpp.

Examples

```
f <- linfun(function(x,y,seg,tp) { x+y }, simplenet)
plot(f)
X <- runiflpp(3, simplenet)
plot(X, add=TRUE, cex=2)
f(X)
```

Linhom *L-function*

Description

Calculates an estimate of the inhomogeneous version of the L-function (Besag's transformation of Ripley's K-function) for a spatial point pattern.

Usage

```
Linhom(...)
```

Arguments

...	Arguments passed to Kinhom to estimate the inhomogeneous K-function.

Details

This command computes an estimate of the inhomogeneous version of the L-function for a spatial point pattern

The original L-function is a transformation (proposed by Besag) of Ripley's K-function,

$$L(r) = \sqrt{\frac{K(r)}{\pi}}$$

where $K(r)$ is the Ripley K-function of a spatially homogeneous point pattern, estimated by Kest.

The inhomogeneous L-function is the corresponding transformation of the inhomogeneous K-function, estimated by Kinhom. It is appropriate when the point pattern clearly does not have a homogeneous intensity of points. It was proposed by Baddeley, Møller and Waagepetersen (2000).

The command Linhom first calls Kinhom to compute the estimate of the inhomogeneous K-function, and then applies the square root transformation.

For a Poisson point pattern (homogeneous or inhomogeneous), the theoretical value of the inhomogeneous L-function is $L(r) = r$. The square root also has the effect of stabilising the variance of the estimator, so that L is more appropriate for use in simulation envelopes and hypothesis tests.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Essentially a data frame containing columns

r	the vector of values of the argument r at which the function L has been estimated
theo	the theoretical value $L(r) = r$ for a stationary Poisson process

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $L(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Møller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

See Also

Kest, Lest, Kinhom, pcf

Examples

```
data(japanesepines)
X <- japanesepines
L <- Linhom(X, sigma=0.1)
plot(L, main="Inhomogeneous L function for Japanese Pines")
```

---

linim *Create Pixel Image on Linear Network*

Description

Creates an object of class "linim" that represents a pixel image on a linear network.

Usage

```
linim(L, Z, ..., restrict=TRUE, df=NULL)
```

Arguments

L	Linear network (object of class "linnet").
Z	Pixel image (object of class "im").
...	Ignored.
restrict	Advanced use only. Logical value indicating whether to ensure that all pixels in Z which do not lie on the network L have pixel value NA. This condition must be satisfied, but if you set restrict=FALSE it will not be checked, and the code will run faster.
df	Advanced use only. Data frame giving full details of the mapping between the pixels of Z and the lines of L. See Details.

Details

This command creates an object of class "linim" that represents a pixel image defined on a linear network. Typically such objects are used to represent the result of smoothing or model-fitting on the network. Most users will not need to call linim directly.

The argument L is a linear network (object of class "linnet"). It gives the exact spatial locations of the line segments of the network, and their connectivity.

The argument Z is a pixel image object of class "im" that gives a pixellated approximation of the function values.

For increased efficiency, advanced users may specify the optional argument df. This is a data frame giving the precomputed mapping between the pixels of Z and the line segments of L. It should have columns named xc, yc containing the coordinates of the pixel centres, x,y containing the projections of these pixel centres onto the linear network, mapXY identifying the line segment on which each projected point lies, and tp giving the parametric position of (x,y) along the segment.

Value

Object of class "linim" that also inherits the class "im". There is a special method for plotting this class.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Ang, Q.W. (2010) *Statistical methodology for events on a network.* Master's thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

McSwiggan, G., Nair, M.G. and Baddeley, A. (2012) Fitting Poisson point process models to events on a linear network. Manuscript in preparation.

See Also

plot.linim, linnet, eval.linim, Math.linim, im.

Examples

```
Z <- as.im(function(x,y) {x-y}, Frame(simplenet))
X <- linim(simplenet, Z)
X
```

---

linnet *Create a Linear Network*

Description

Creates an object of class "linnet" representing a network of line segments.

Usage

```
linnet(vertices, m, edges, sparse=FALSE, warn=TRUE)
```

Arguments

vertices	Point pattern (object of class "ppp") specifying the vertices of the network.
m	Adjacency matrix. A matrix or sparse matrix of logical values equal to TRUE when the corresponding vertices are joined by a line. (Specify either m or edges.)
edges	Edge list. A two-column matrix of integers, specifying all pairs of vertices that should be joined by an edge. (Specify either m or edges.)
sparse	Optional. Logical value indicating whether to use a sparse matrix representation of the network. See Details.
warn	Logical value indicating whether to issue a warning if the resulting network is not connected.

Details

An object of class "linnet" represents a network of straight line segments in two dimensions. The function linnet creates such an object from the minimal information: the spatial location of each vertex (endpoint, crossing point or meeting point of lines) and information about which vertices are joined by an edge.

If sparse=FALSE (the default), the algorithm will compute and store various properties of the network, including the adjacency matrix m and a matrix giving the shortest-path distances between each pair of vertices in the network. This is more efficient for small datasets. However it can require large amounts of memory and can take a long time to execute.

If sparse=TRUE, then the shortest-path distances will not be computed, and the network adjacency matrix m will be stored as a sparse matrix. This saves a lot of time and memory when creating the linear network.

If the argument edges is given, then it will also determine the *ordering* of the line segments when they are stored or extracted. For example, edges[i,] corresponds to as.psp(L)[i].

Value

Object of class "linnet" representing the linear network.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

`simplenet` for an example of a linear network.

`methods.linnet` for methods applicable to `linnet` objects.

Special tools: `thinNetwork`, `insertVertices`, `connected.linnet`, `lixellate`.

`delaunayNetwork` for the Delaunay triangulation as a network.

`ppp`, `psp`.

Examples

```
# letter 'A' specified by adjacency matrix
v <- ppp(x=(-2):2, y=3*c(0,1,2,1,0), c(-3,3), c(-1,7))
m <- matrix(FALSE, 5,5)
for(i in 1:4) m[i,i+1] <- TRUE
m[2,4] <- TRUE
m <- m | t(m)
letterA <- linnet(v, m)
plot(letterA)

# letter 'A' specified by edge list
edg <- cbind(1:4, 2:5)
edg <- rbind(edg, c(2,4))
letterA <- linnet(v, edges=edg)
```

---

lintess *Tessellation on a Linear Network*

Description

Create a tessellation on a linear network.

Usage

```
lintess(L, df)
```

Arguments

`L`	Linear network (object of class `"linnet"`).
`df`	Data frame of coordinates of endpoints of the tiles of the tessellation.

Details

A tessellation on a linear network `L` is a partition of the network into non-overlapping pieces (tiles). Each tile consists of one or more line segments which are subsets of the line segments making up the network. A tile can consist of several disjoint pieces.

The data frame `df` should have columns named `seg`, `t0`, `t1` and `tile`.

Each row of the data frame specifies one sub-segment of the network and allocates it to a particular tile.

The seg column specifies which line segment of the network contains the sub-segment. Values of seg are integer indices for the segments in as.psp(L).

The t0 and t1 columns specify the start and end points of the sub-segment. They should be numeric values between 0 and 1 inclusive, where the values 0 and 1 representing the network vertices that are joined by this network segment.

The tile column specifies which tile of the tessellation includes this sub-segment. It will be coerced to a factor and its levels will be the names of the tiles.

If df is missing or NULL, the result is a tessellation with only one tile, consisting of the entire network L.

Value

An object of class "lintess". There are methods for print, plot and summary for this object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Greg McSwiggan.

See Also

linnet for linear networks.

plot.lintess for plotting.

divide.linnet to make a tessellation demarcated by given points.

lineardirichlet to create the Dirichlet-Voronoi tessellation from a point pattern on a linear network.

as.linfun.lintess, as.linnet.lintess and as.linim to convert to other classes.

tile.lengths to compute the length of each tile in the tessellation.

The undocumented methods Window.lintess and as.owin.lintess extract the spatial window.

Examples

```
# tessellation consisting of one tile for each existing segment
ns <- nsegments(simplenet)
df <- data.frame(seg=1:ns, t0=0, t1=1, tile=letters[1:ns])
u <- lintess(simplenet, df)
u
plot(u)
```

---

lixellate *Subdivide Segments of a Network*

Description

Each line segment of a linear network will be divided into several shorter segments (line elements or lixels).

Usage

```
lixellate(X, ..., nsplit, eps, sparse = TRUE)
```

Arguments

- `X` — A linear network (object of class "linnet") or a point pattern on a linear network (object of class "lpp").
- `...` — Ignored.
- `nsplit` — Number of pieces into which *each* line segment of X should be divided. Either a single integer, or an integer vector with one entry for each line segment in X. Incompatible with eps.
- `eps` — Maximum length of the resulting pieces of line segment. A single numeric value. Incompatible with nsplit.
- `sparse` — Optional. Logical value specifying whether the resulting linear network should be represented using a sparse matrix. If sparse=NULL, then the representation will be the same as in X.

Details

Each line segment in X will be subdivided into equal pieces. The result is an object of the same kind as X, representing the same data as X except that the segments have been subdivided.

Splitting is controlled by the arguments nsplit and eps, exactly one of which should be given.

If nsplit is given, it specifies the number of pieces into which *each* line segment of X should be divided. It should be either a single integer, or an integer vector of length equal to the number of line segments in X.

If eps is given, it specifies the maximum length of any resulting piece of line segment.

It is strongly advisable to use sparse=TRUE (the default) to limit the computation time.

If X is a point pattern (class "lpp") then the spatial coordinates and marks of each data point are unchanged, but the local coordinates will change, because they are adjusted to map them to the new subdivided network.

Value

Object of the same kind as X.

Author(s)

Greg McSwiggan, Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

linnet, lpp.

Examples

```
A <- lixellate(simplenet, nsplit=4)
plot(A, main="lixellate(simplenet, nsplit=4)")
points(vertices(A), pch=16)

spiders
lixellate(spiders, nsplit=3)
```

localK *Neighbourhood density function*

Description

Computes the neighbourhood density function, a local version of the K-function or L-function, defined by Getis and Franklin (1987).

Usage

```
localK(X, ..., correction = "Ripley", verbose = TRUE, rvalue=NULL)
localL(X, ..., correction = "Ripley", verbose = TRUE, rvalue=NULL)
```

Arguments

`X`	A point pattern (object of class "ppp").
`...`	Ignored.
`correction`	String specifying the edge correction to be applied. Options are "none", "translate", "translation", "Ripley", "isotropic" or "best". Only one correction may be specified.
`verbose`	Logical flag indicating whether to print progress reports during the calculation.
`rvalue`	Optional. A *single* value of the distance argument r at which the function L or K should be computed.

Details

The command `localL` computes the *neighbourhood density function*, a local version of the L-function (Besag's transformation of Ripley's K-function) that was proposed by Getis and Franklin (1987). The command `localK` computes the corresponding local analogue of the K-function.

Given a spatial point pattern `X`, the neighbourhood density function $L_i(r)$ associated with the ith point in `X` is computed by

$$L_i(r) = \sqrt{\frac{a}{(n-1)\pi} \sum_j e_{ij}}$$

where the sum is over all points $j \neq i$ that lie within a distance r of the ith point, a is the area of the observation window, n is the number of points in `X`, and e_{ij} is an edge correction term (as described in `Kest`). The value of $L_i(r)$ can also be interpreted as one of the summands that contributes to the global estimate of the L function.

By default, the function $L_i(r)$ or $K_i(r)$ is computed for a range of r values for each point i. The results are stored as a function value table (object of class "fv") with a column of the table containing the function estimates for each point of the pattern `X`.

Alternatively, if the argument `rvalue` is given, and it is a single number, then the function will only be computed for this value of r, and the results will be returned as a numeric vector, with one entry of the vector for each point of the pattern `X`.

Inhomogeneous counterparts of `localK` and `localL` are computed by `localKinhom` and `localLinhom`.

Value

If rvalue is given, the result is a numeric vector of length equal to the number of points in the point pattern.

If rvalue is absent, the result is an object of class "fv", see fv.object, which can be plotted directly using plot.fv. Essentially a data frame containing columns

r	the vector of values of the argument r at which the function K has been estimated
theo	the theoretical value $K(r) = \pi r^2$ or $L(r) = r$ for a stationary Poisson process

together with columns containing the values of the neighbourhood density function for each point in the pattern. Column i corresponds to the ith point. The last two columns contain the r and theo values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Getis, A. and Franklin, J. (1987) Second-order neighbourhood analysis of mapped point patterns. *Ecology* **68**, 473–477.

See Also

Kest, Lest, localKinhom, localLinhom.

Examples

```
data(ponderosa)
X <- ponderosa

# compute all the local L functions
L <- localL(X)

# plot all the local L functions against r
plot(L, main="local L functions for ponderosa", legend=FALSE)

# plot only the local L function for point number 7
plot(L, iso007 ~ r)

# compute the values of L(r) for r = 12 metres
L12 <- localL(X, rvalue=12)

# Spatially interpolate the values of L12
# Compare Figure 5(b) of Getis and Franklin (1987)
X12 <- X %mark% L12
Z <- Smooth(X12, sigma=5, dimyx=128)

plot(Z, col=topo.colors(128), main="smoothed neighbourhood density")
contour(Z, add=TRUE)
points(X, pch=16, cex=0.5)
```

localKinhom *Inhomogeneous Neighbourhood Density Function*

Description

Computes spatially-weighted versions of the the local K-function or L-function.

Usage

```
localKinhom(X, lambda, ...,
            correction = "Ripley", verbose = TRUE, rvalue=NULL,
            sigma = NULL, varcov = NULL)
localLinhom(X, lambda, ...,
            correction = "Ripley", verbose = TRUE, rvalue=NULL,
            sigma = NULL, varcov = NULL)
```

Arguments

X	A point pattern (object of class "ppp").
lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
...	Extra arguments. Ignored if lambda is present. Passed to density.ppp if lambda is omitted.
correction	String specifying the edge correction to be applied. Options are "none", "translate", "Ripley", "translation", "isotropic" or "best". Only one correction may be specified.
verbose	Logical flag indicating whether to print progress reports during the calculation.
rvalue	Optional. A *single* value of the distance argument r at which the function L or K should be computed.
sigma, varcov	Optional arguments passed to density.ppp to control the kernel smoothing procedure for estimating lambda, if lambda is missing.

Details

The functions localKinhom and localLinhom are inhomogeneous or weighted versions of the neighbourhood density function implemented in localK and localL.

Given a spatial point pattern X, the inhomogeneous neighbourhood density function $L_i(r)$ associated with the ith point in X is computed by

$$L_i(r) = \sqrt{\frac{1}{\pi} \sum_j \frac{e_{ij}}{\lambda_j}}$$

where the sum is over all points $j \neq i$ that lie within a distance r of the ith point, λ_j is the estimated intensity of the point pattern at the point j, and e_{ij} is an edge correction term (as described in Kest). The value of $L_i(r)$ can also be interpreted as one of the summands that contributes to the global estimate of the inhomogeneous L function (see Linhom).

By default, the function $L_i(r)$ or $K_i(r)$ is computed for a range of r values for each point i. The results are stored as a function value table (object of class "fv") with a column of the table containing the function estimates for each point of the pattern X.

Alternatively, if the argument rvalue is given, and it is a single number, then the function will only be computed for this value of r, and the results will be returned as a numeric vector, with one entry of the vector for each point of the pattern X.

Value

If rvalue is given, the result is a numeric vector of length equal to the number of points in the point pattern.

If rvalue is absent, the result is an object of class "fv", see fv.object, which can be plotted directly using plot.fv. Essentially a data frame containing columns

r	the vector of values of the argument r at which the function K has been estimated
theo	the theoretical value $K(r) = \pi r^2$ or $L(r) = r$ for a stationary Poisson process

together with columns containing the values of the neighbourhood density function for each point in the pattern. Column i corresponds to the ith point. The last two columns contain the r and theo values.

Author(s)

Mike Kuhn, Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Kinhom, Linhom, localK, localL.

Examples

```
data(ponderosa)
X <- ponderosa

# compute all the local L functions
L <- localLinhom(X)

# plot all the local L functions against r
plot(L, main="local L functions for ponderosa", legend=FALSE)

# plot only the local L function for point number 7
plot(L, iso007 ~ r)

# compute the values of L(r) for r = 12 metres
L12 <- localL(X, rvalue=12)
```

localpcf *Local pair correlation function*

Description

Computes individual contributions to the pair correlation function from each data point.

Usage

```
localpcf(X, ..., delta=NULL, rmax=NULL, nr=512, stoyan=0.15)
localpcfinhom(X, ..., delta=NULL, rmax=NULL, nr=512, stoyan=0.15,
       lambda=NULL, sigma=NULL, varcov=NULL)
```

Arguments

X	A point pattern (object of class "ppp").
delta	Smoothing bandwidth for pair correlation. The halfwidth of the Epanechnikov kernel.
rmax	Optional. Maximum value of distance r for which pair correlation values $g(r)$ should be computed.
nr	Optional. Number of values of distance r for which pair correlation $g(r)$ should be computed.
stoyan	Optional. The value of the constant c in Stoyan's rule of thumb for selecting the smoothing bandwidth delta.
lambda	Optional. Values of the estimated intensity function, for the inhomogeneous pair correlation. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
sigma,varcov,...	These arguments are ignored by localpcf but are passed by localpcfinhom (when lambda=NULL) to the function density.ppp to control the kernel smoothing estimation of lambda.

Details

localpcf computes the contribution, from each individual data point in a point pattern X, to the empirical pair correlation function of X. These contributions are sometimes known as LISA (local indicator of spatial association) functions based on pair correlation.

localpcfinhom computes the corresponding contribution to the *inhomogeneous* empirical pair correlation function of X.

Given a spatial point pattern X, the local pcf $g_i(r)$ associated with the ith point in X is computed by

$$g_i(r) = \frac{a}{2\pi n} \sum_j k(d_{i,j} - r)$$

where the sum is over all points $j \neq i$, a is the area of the observation window, n is the number of points in X, and d_{ij} is the distance between points i and j. Here k is the Epanechnikov kernel,

$$k(t) = \frac{3}{4\delta} \max(0, 1 - \frac{t^2}{\delta^2}).$$

Edge correction is performed using the border method (for the sake of computational efficiency): the estimate $g_i(r)$ is set to NA if $r > b_i$, where b_i is the distance from point i to the boundary of the observation window.

The smoothing bandwidth δ may be specified. If not, it is chosen by Stoyan's rule of thumb $\delta = c/\hat{\lambda}$ where $\hat{\lambda} = n/a$ is the estimated intensity and c is a constant, usually taken to be 0.15. The value of c is controlled by the argument stoyan.

For localpcfinhom, the optional argument lambda specifies the values of the estimated intensity function. If lambda is given, it should be either a numeric vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(x,y) which can be evaluated to give the intensity value at any location. If lambda is not given, then it will be estimated using a leave-one-out kernel density smoother as described in pcfinhom.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv. Essentially a data frame containing columns

r	the vector of values of the argument r at which the function K has been estimated
theo	the theoretical value $K(r) = \pi r^2$ or $L(r) = r$ for a stationary Poisson process

together with columns containing the values of the local pair correlation function for each point in the pattern. Column i corresponds to the ith point. The last two columns contain the r and theo values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

localK, localKinhom, pcf, pcfinhom

Examples

```
data(ponderosa)
X <- ponderosa

g <- localpcf(X, stoyan=0.5)
colo <- c(rep("grey", npoints(X)), "blue")
a <- plot(g, main=c("local pair correlation functions", "Ponderosa pines"),
        legend=FALSE, col=colo, lty=1)

# plot only the local pair correlation function for point number 7
plot(g, est007 ~ r)

gi <- localpcfinhom(X, stoyan=0.5)
a <- plot(gi, main=c("inhomogeneous local pair correlation functions",
                     "Ponderosa pines"),
              legend=FALSE, col=colo, lty=1)
```

logLik.dppm *Log Likelihood and AIC for Fitted Determinantal Point Process Model*

Description

Extracts the log Palm likelihood, deviance, and AIC of a fitted determinantal point process model.

Usage

```
## S3 method for class 'dppm'
logLik(object, ...)
## S3 method for class 'dppm'
AIC(object, ..., k=2)
## S3 method for class 'dppm'
extractAIC(fit, scale=0, k=2, ...)
## S3 method for class 'dppm'
nobs(object, ...)
```

Arguments

object,fit	Fitted point process model. An object of class "dppm".
...	Ignored.
scale	Ignored.
k	Numeric value specifying the weight of the equivalent degrees of freedom in the AIC. See Details.

Details

These functions are methods for the generic commands `logLik`, `extractAIC` and `nobs` for the class "dppm".

An object of class "dppm" represents a fitted Cox or cluster point process model. It is obtained from the model-fitting function `dppm`.

These methods apply only when the model was fitted by maximising the Palm likelihood (Tanaka et al, 2008) by calling `dppm` with the argument `method="palm"`.

The method `logLik.dppm` computes the maximised value of the log Palm likelihood for the fitted model `object`.

The methods `AIC.dppm` and `extractAIC.dppm` compute the Akaike Information Criterion AIC for the fitted model based on the Palm likelihood (Tanaka et al, 2008)

$$AIC = -2\log(PL) + k \times \text{edf}$$

where PL is the maximised Palm likelihood of the fitted model, and edf is the effective degrees of freedom of the model.

The method `nobs.dppm` returns the number of points in the original data point pattern to which the model was fitted.

The R function `step` uses these methods, but it does not work for determinantal models yet due to a missing implementation of `update.dppm`.

Value

logLik returns a numerical value, belonging to the class "logLik", with an attribute "df" giving the degrees of freedom.

AIC returns a numerical value.

extractAIC returns a numeric vector of length 2 containing the degrees of freedom and the AIC value.

nobs returns an integer value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Tanaka, U. and Ogata, Y. and Stoyan, D. (2008) Parameter estimation and model selection for Neyman-Scott point processes. *Biometrical Journal* **50**, 43–57.

See Also

dppm, logLik.ppm

Examples

```
fit <- dppm(swedishpines ~ x, dppGauss(), method="palm")
nobs(fit)
logLik(fit)
extractAIC(fit)
AIC(fit)
```

---

logLik.kppm *Log Likelihood and AIC for Fitted Cox or Cluster Point Process Model*

Description

Extracts the log Palm likelihood, deviance, and AIC of a fitted Cox or cluster point process model.

Usage

```
## S3 method for class 'kppm'
logLik(object, ...)
## S3 method for class 'kppm'
AIC(object, ..., k=2)
## S3 method for class 'kppm'
extractAIC(fit, scale=0, k=2, ...)
## S3 method for class 'kppm'
nobs(object, ...)
```

Arguments

`object,fit`	Fitted point process model. An object of class "kppm".
`...`	Ignored.
`scale`	Ignored.
`k`	Numeric value specifying the weight of the equivalent degrees of freedom in the AIC. See Details.

Details

These functions are methods for the generic commands `logLik`, `extractAIC` and `nobs` for the class "kppm".

An object of class "kppm" represents a fitted Cox or cluster point process model. It is obtained from the model-fitting function `kppm`.

These methods apply only when the model was fitted by maximising the Palm likelihood (Tanaka et al, 2008) by calling `kppm` with the argument `method="palm"`.

The method `logLik.kppm` computes the maximised value of the log Palm likelihood for the fitted model `object`.

The methods `AIC.kppm` and `extractAIC.kppm` compute the Akaike Information Criterion AIC for the fitted model based on the Palm likelihood (Tanaka et al, 2008)

$$AIC = -2\log(PL) + k \times \text{edf}$$

where PL is the maximised Palm likelihood of the fitted model, and edf is the effective degrees of freedom of the model.

The method `nobs.kppm` returns the number of points in the original data point pattern to which the model was fitted.

The R function `step` uses these methods.

Value

`logLik` returns a numerical value, belonging to the class "logLik", with an attribute "df" giving the degrees of freedom.

`AIC` returns a numerical value.

`extractAIC` returns a numeric vector of length 2 containing the degrees of freedom and the AIC value.

`nobs` returns an integer value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Tanaka, U. and Ogata, Y. and Stoyan, D. (2008) Parameter estimation and model selection for Neyman-Scott point processes. *Biometrical Journal* **50**, 43–57.

See Also

kppm, logLik.ppm

Examples

```
fit <- kppm(redwood ~ x, "Thomas", method="palm")
nobs(fit)
logLik(fit)
extractAIC(fit)
AIC(fit)
step(fit)
```

---

logLik.mppm — *Log Likelihood and AIC for Multiple Point Process Model*

Description

For a point process model that has been fitted to multiple point patterns, these functions extract the log likelihood and AIC, or analogous quantities based on the pseudolikelihood.

Usage

```
## S3 method for class 'mppm'
logLik(object, ..., warn=TRUE)

## S3 method for class 'mppm'
AIC(object, ..., k=2, takeuchi=TRUE)

## S3 method for class 'mppm'
extractAIC(fit, scale = 0, k = 2, ..., takeuchi = TRUE)

## S3 method for class 'mppm'
nobs(object, ...)

## S3 method for class 'mppm'
getCall(x, ...)

## S3 method for class 'mppm'
terms(x, ...)
```

Arguments

object,fit,x	Fitted point process model (fitted to multiple point patterns). An object of class "mppm".
...	Ignored.
warn	If TRUE, a warning is given when the pseudolikelihood is returned instead of the likelihood.
scale	Ignored.
k	Numeric value specifying the weight of the equivalent degrees of freedom in the AIC. See Details.
takeuchi	Logical value specifying whether to use the Takeuchi penalty (takeuchi=TRUE) or the number of fitted parameters (takeuchi=FALSE) in calculating AIC.

Details

These functions are methods for the generic commands logLik, AIC, extractAIC, terms and getCall for the class "mppm".

An object of class "mppm" represents a fitted Poisson or Gibbs point process model fitted to several point patterns. It is obtained from the model-fitting function mppm.

The method logLik.mppm extracts the maximised value of the log likelihood for the fitted model (as approximated by quadrature using the Berman-Turner approximation). If object is not a Poisson process, the maximised log *pseudolikelihood* is returned, with a warning.

The Akaike Information Criterion AIC for a fitted model is defined as

$$AIC = -2\log(L) + k \times \text{penalty}$$

where L is the maximised likelihood of the fitted model, and penalty is a penalty for model complexity, usually equal to the effective degrees of freedom of the model. The method extractAIC.mppm returns the *analogous* quantity $AIC*$ in which L is replaced by $L*$, the quadrature approximation to the likelihood (if fit is a Poisson model) or the pseudolikelihood (if fit is a Gibbs model).

The penalty term is calculated as follows. If takeuchi=FALSE then penalty is the number of fitted parameters. If takeuchi=TRUE then penalty $= \text{trace}(JH^{-1})$ where J and H are the estimated variance and hessian, respectively, of the composite score. These two choices are equivalent for a Poisson process.

The method nobs.mppm returns the total number of points in the original data point patterns to which the model was fitted.

The method getCall.mppm extracts the original call to mppm which caused the model to be fitted.

The method terms.mppm extracts the covariate terms in the model formula as a terms object. Note that these terms do not include the interaction component of the model.

The R function step uses these methods.

Value

See the help files for the corresponding generic functions.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curti
Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

See Also

mppm

Examples

```
fit <- mppm(Bugs ~ x, hyperframe(Bugs=waterstriders))
logLik(fit)
AIC(fit)
nobs(fit)
getCall(fit)
```

logLik.ppm *Log Likelihood and AIC for Point Process Model*

Description

Extracts the log likelihood, deviance, and AIC of a fitted Poisson point process model, or analogous quantities based on the pseudolikelihood or logistic likelihood for a fitted Gibbs point process model.

Usage

```
## S3 method for class 'ppm'
logLik(object, ..., new.coef=NULL, warn=TRUE, absolute=FALSE)

## S3 method for class 'ppm'
deviance(object, ...)

## S3 method for class 'ppm'
AIC(object, ..., k=2, takeuchi=TRUE)

## S3 method for class 'ppm'
extractAIC(fit, scale=0, k=2, ..., takeuchi=TRUE)

## S3 method for class 'ppm'
nobs(object, ...)
```

Arguments

`object,fit`	Fitted point process model. An object of class "ppm".
`...`	Ignored.
`warn`	If `TRUE`, a warning is given when the pseudolikelihood or logistic likelihood is returned instead of the likelihood.
`absolute`	Logical value indicating whether to include constant terms in the loglikelihood.
`scale`	Ignored.
`k`	Numeric value specifying the weight of the equivalent degrees of freedom in the AIC. See Details.
`new.coef`	New values for the canonical parameters of the model. A numeric vector of the same length as `coef(object)`.
`takeuchi`	Logical value specifying whether to use the Takeuchi penalty (`takeuchi=TRUE`) or the number of fitted parameters (`takeuchi=FALSE`) in calculating AIC.

Details

These functions are methods for the generic commands `logLik`, `deviance`, `extractAIC` and `nobs` for the class "ppm".

An object of class "ppm" represents a fitted Poisson or Gibbs point process model. It is obtained from the model-fitting function `ppm`.

The method `logLik.ppm` computes the maximised value of the log likelihood for the fitted model `object` (as approximated by quadrature using the Berman-Turner approximation) is extracted. If object is not a Poisson process, the maximised log *pseudolikelihood* is returned, with a warning (if warn=TRUE).

The Akaike Information Criterion AIC for a fitted model is defined as

$$AIC = -2\log(L) + k \times \text{penalty}$$

where L is the maximised likelihood of the fitted model, and penalty is a penalty for model complexity, usually equal to the effective degrees of freedom of the model. The method extractAIC.ppm returns the *analogous* quantity $AIC*$ in which L is replaced by $L*$, the quadrature approximation to the likelihood (if fit is a Poisson model) or the pseudolikelihood or logistic likelihood (if fit is a Gibbs model).

The penalty term is calculated as follows. If takeuchi=FALSE then penalty is the number of fitted parameters. If takeuchi=TRUE then penalty $=$ trace(JH^{-1}) where J and H are the estimated variance and hessian, respectively, of the composite score. These two choices are equivalent for a Poisson process.

The method nobs.ppm returns the number of points in the original data point pattern to which the model was fitted.

The R function step uses these methods.

Value

logLik returns a numerical value, belonging to the class "logLik", with an attribute "df" giving the degrees of freedom.

AIC returns a numerical value.

extractAIC returns a numeric vector of length 2 containing the degrees of freedom and the AIC value.

nobs returns an integer value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Varin, C. and Vidoni, P. (2005) A note on composite likelihood inference and model selection. *Biometrika* **92**, 519–528.

See Also

ppm, as.owin, coef.ppm, fitted.ppm, formula.ppm, model.frame.ppm, model.matrix.ppm, plot.ppm, predict.ppm, residuals.ppm, simulate.ppm, summary.ppm, terms.ppm, update.ppm, vcov.ppm.

Examples

```
data(cells)
fit <- ppm(cells, ~x)
nobs(fit)
logLik(fit)
deviance(fit)
```

```
extractAIC(fit)
AIC(fit)
step(fit)
```

logLik.slrm *Loglikelihood of Spatial Logistic Regression*

Description

Computes the (maximised) loglikelihood of a fitted Spatial Logistic Regression model.

Usage

```
## S3 method for class 'slrm'
logLik(object, ..., adjust = TRUE)
```

Arguments

object	a fitted spatial logistic regression model. An object of class "slrm".
...	Ignored.
adjust	Logical value indicating whether to adjust the loglikelihood of the model to make it comparable with a point process likelihood. See Details.

Details

This is a method for logLik for fitted spatial logistic regression models (objects of class "slrm", usually obtained from the function slrm). It computes the log-likelihood of a fitted spatial logistic regression model.

If adjust=FALSE, the loglikelihood is computed using the standard formula for the loglikelihood of a logistic regression model for a finite set of (pixel) observations.

If adjust=TRUE then the loglikelihood is adjusted so that it is approximately comparable with the likelihood of a point process in continuous space, by subtracting the value $n \log(a)$ where n is the number of points in the original point pattern dataset, and a is the area of one pixel.

Value

A numerical value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

slrm

Examples

```
X <- rpoispp(42)
fit <- slrm(X ~ x+y)
logLik(fit)
logLik(fit, adjust=FALSE)
```

lohboot *Bootstrap Confidence Bands for Summary Function*

Description

Computes a bootstrap confidence band for a summary function of a point process.

Usage

```
lohboot(X,
        fun=c("pcf", "Kest", "Lest", "pcfinhom", "Kinhom", "Linhom"),
        ...,
         block=FALSE, global=FALSE, basicboot=FALSE, Vcorrection=FALSE,
         confidence=0.95, nx = 4, ny = nx, nsim=200, type=7)
```

Arguments

`X`	A point pattern (object of class "ppp").
`fun`	Name of the summary function for which confidence intervals are desired: one of the strings "pcf", "Kest", "Lest", "pcfinhom", "Kinhom" or "Linhom". Alternatively, the function itself; it must be one of the functions listed here.
`...`	Arguments passed to the corresponding local version of the summary function (see Details).
`block`	Logical value indicating whether to use Loh's block bootstrap as originally proposed. Default is `FALSE` for consistency with older code. See Details.
`global`	Logical. If `FALSE` (the default), pointwise confidence intervals are constructed. If `TRUE`, a global (simultaneous) confidence band is constructed.
`basicboot`	Logical value indicating whether to use the so-called basic bootstrap confidence interval. See Details.
`Vcorrection`	Logical value indicating whether to use a variance correction when `fun="Kest"` or `fun="Kinhom"`. See Details.
`confidence`	Confidence level, as a fraction between 0 and 1.
`nx,ny`	Integers. If `block=TRUE`, divide the window into `nx*ny` rectangles.
`nsim`	Number of bootstrap simulations.
`type`	Integer. Type of quantiles. Argument passed to `quantile.default` controlling the way the quantiles are calculated.

Details

This algorithm computes confidence bands for the true value of the summary function `fun` using the bootstrap method of Loh (2008) and a modification described in Baddeley, Rubak, Turner (2015).

If `fun="pcf"`, for example, the algorithm computes a pointwise `(100 * confidence)`% confidence interval for the true value of the pair correlation function for the point process, normally estimated by `pcf`. It starts by computing the array of *local* pair correlation functions, `localpcf`, of the data pattern `X`. This array consists of the contributions to the estimate of the pair correlation function from each data point.

If `block=FALSE`, these contributions are resampled `nsim` times with replacement as described in Baddeley, Rubak, Turner (2015); from each resampled dataset the total contribution is computed, yielding `nsim` random pair correlation functions.

If `block=TRUE`, the (bounding box of the) window is divided into $nx * ny$ rectangles (blocks). The average contribution of a block is obtained by averaging the contribution of each point included in the block. Then, the average contributions on each block are resampled `nsim` times with replacement as described in Loh (2008) and Loh (2010); from each resampled dataset the total contribution is computed, yielding `nsim` random pair correlation functions. Notice that for non-rectangular windows any blocks not fully contained in the window are discarded before doing the resampling, so the effective number of blocks may be substantially smaller than $nx * ny$ in this case.

The pointwise `alpha/2` and `1 - alpha/2` quantiles of these functions are computed, where `alpha = 1 - confidence`. The average of the local functions is also computed as an estimate of the pair correlation function.

There are several ways to define a bootstrap confidence interval. If `basicbootstrap=TRUE`, the so-called basic confidence bootstrap interval is used as described in Loh (2008).

It has been noticed in Loh (2010) that when the intensity of the point process is unknown, the bootstrap error estimate is larger than it should be. When the K function is used, an adjustment procedure has been proposed in Loh (2010) that is used if `Vcorrection=TRUE`. In this case, the basic confidence bootstrap interval is implicitly used.

To control the estimation algorithm, use the arguments `...`, which are passed to the local version of the summary function, as shown below:

fun	**local version**
`pcf`	`localpcf`
`Kest`	`localK`
`Lest`	`localK`
`pcfinhom`	`localpcfinhom`
`Kinhom`	`localKinhom`
`Linhom`	`localKinhom`

For `fun="Lest"`, the calculations are first performed as if `fun="Kest"`, and then the square-root transformation is applied to obtain the L-function.

Note that the confidence bands computed by `lohboot(fun="pcf")` may not contain the estimate of the pair correlation function computed by `pcf`, because of differences between the algorithm parameters (such as the choice of edge correction) in `localpcf` and `pcf`. If you are using `lohboot`, the appropriate point estimate of the pair correlation itself is the pointwise mean of the local estimates, which is provided in the result of `lohboot` and is shown in the default plot.

If the confidence bands seem unbelievably narrow, this may occur because the point pattern has a hard core (the true pair correlation function is zero for certain values of distance) or because of an optical illusion when the function is steeply sloping (remember the width of the confidence bands should be measured *vertically*).

An alternative to `lohboot` is `varblock`.

Value

A function value table (object of class `"fv"`) containing columns giving the estimate of the summary function, the upper and lower limits of the bootstrap confidence interval, and the theoretical value of the summary function for a Poisson process.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> and Christophe Biscio.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

Loh, J.M. (2008) A valid and fast spatial bootstrap for correlation functions. *The Astrophysical Journal*, **681**, 726–734.

Loh, J.M. (2010) Bootstrapping an inhomogeneous point process. *Journal of Statistical Planning and Inference*, **140**, 734–749.

See Also

Summary functions `Kest`, `pcf`, `Kinhom`, `pcfinhom`, `localK`, `localpcf`, `localKinhom`, `localpcfinhom`.

See `varblock` for an alternative bootstrap technique.

Examples

```
p <- lohboot(simdat, stoyan=0.5)
plot(p)
```

---

lpp — *Create Point Pattern on Linear Network*

Description

Creates an object of class "`lpp`" that represents a point pattern on a linear network.

Usage

```
lpp(X, L, ...)
```

Arguments

`X`	Locations of the points. A matrix or data frame of coordinates, or a point pattern object (of class "`ppp`") or other data acceptable to `as.ppp`.
`L`	Linear network (object of class "`linnet`").
`...`	Ignored.

Details

This command creates an object of class "`lpp`" that represents a point pattern on a linear network.

Normally `X` is a point pattern. The points of `X` should lie on the lines of `L`.

Alternatively `X` may be a matrix or data frame containing at least two columns.

- Usually the first two columns of `X` will be interpreted as spatial coordinates, and any remaining columns as marks.

- An exception occurs if X is a data frame with columns named x, y, seg and tp. Then x and y will be interpreted as spatial coordinates, and seg and tp as local coordinates, with seg indicating which line segment of L the point lies on, and tp indicating how far along the segment the point lies (normalised to 1). Any remaining columns will be interpreted as marks.
- Another exception occurs if X is a data frame with columns named seg and tp. Then seg and tp will be interpreted as local coordinates, as above, and the spatial coordinates x,y will be computed from them. Any remaining columns will be interpreted as marks.

If X is missing or NULL, the result is an empty point pattern (i.e. containing no points).

Value

An object of class "lpp". Also inherits the class "ppx".

Note on changed format

The internal format of "lpp" objects was changed in **spatstat** version 1.28-0. Objects in the old format are still handled correctly, but computations are faster in the new format. To convert an object X from the old format to the new format, use X <- lpp(as.ppp(X), as.linnet(X)).

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

Installed datasets which are "lpp" objects: chicago, dendrite, spiders.

See as.lpp for converting data to an lpp object.

See methods.lpp and methods.ppx for other methods applicable to lpp objects.

Calculations on an lpp object: intensity.lpp, distfun.lpp, nndist.lpp, nnwhich.lpp, nncross.lpp, nnfun.lpp.

Summary functions: linearK, linearKinhom, linearpcf, linearKdot, linearKcross, linearmarkconnect, etc.

Random point patterns on a linear network can be generated by rpoislpp or runiflpp.

See linnet for linear networks.

Examples

```
# letter 'A'
v <- ppp(x=(-2):2, y=3*c(0,1,2,1,0), c(-3,3), c(-1,7))
edg <- cbind(1:4, 2:5)
edg <- rbind(edg, c(2,4))
letterA <- linnet(v, edges=edg)

# points on letter A
xx <- list(x=c(-1.5,0,0.5,1.5), y=c(1.5,3,4.5,1.5))
X <- lpp(xx, letterA)

plot(X)
X
summary(X)

# empty pattern
lpp(L=letterA)
```

`lppm` *Fit Point Process Model to Point Pattern on Linear Network*

Description

Fit a point process model to a point pattern dataset on a linear network

Usage

```
lppm(X, ...)

## S3 method for class 'formula'
lppm(X, interaction=NULL, ..., data=NULL)

## S3 method for class 'lpp'
lppm(X, ..., eps=NULL, nd=1000, random=FALSE)
```

Arguments

`X`	Either an object of class `"lpp"` specifying a point pattern on a linear network, or a `formula` specifying the point process model.
`...`	Arguments passed to `ppm`.
`interaction`	An object of class `"interact"` describing the point process interaction structure, or `NULL` indicating that a Poisson process (stationary or nonstationary) should be fitted.
`data`	Optional. The values of spatial covariates (other than the Cartesian coordinates) required by the model. A list whose entries are images, functions, windows, tessellations or single numbers.
`eps`	Optional. Spacing between dummy points along each segment of the network.
`nd`	Optional. Total number of dummy points placed on the network. Ignored if `eps` is given.
`random`	Logical value indicating whether the grid of dummy points should be placed at a randomised starting position.

Details

This function fits a point process model to data that specify a point pattern on a linear network. It is a counterpart of the model-fitting function `ppm` designed to work with objects of class `"lpp"` instead of `"ppp"`.

The function `lppm` is generic, with methods for the classes `formula` and `lppp`.

In `lppm.lpp` the first argument `X` should be an object of class `"lpp"` (created by the command `lpp`) specifying a point pattern on a linear network.

In `lppm.formula`, the first argument is a `formula` in the R language describing the spatial trend model to be fitted. It has the general form `pattern ~ trend` where the left hand side `pattern` is usually the name of a point pattern on a linear network (object of class `"lpp"`) to which the model should be fitted, or an expression which evaluates to such a point pattern; and the right hand side `trend` is an expression specifying the spatial trend of the model.

Other arguments `...` are passed from `lppm.formula` to `lppm.lpp` and from `lppm.lpp` to `ppm`.

Value

An object of class "lppm" representing the fitted model. There are methods for `print`, `predict`, `coef` and similar functions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Greg McSwiggan.

References

Ang, Q.W. (2010) *Statistical methodology for events on a network*. Master's thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

McSwiggan, G., Nair, M.G. and Baddeley, A. (2012) Fitting Poisson point process models to events on a linear network. Manuscript in preparation.

See Also

`methods.lppm`, `predict.lppm`, `ppm`, `lpp`.

Examples

```
X <- runiflpp(15, simplenet)
lppm(X ~1)
lppm(X ~x)
marks(X) <- factor(rep(letters[1:3], 5))
lppm(X ~ marks)
lppm(X ~ marks * x)
```

`lurking` *Lurking Variable Plot*

Description

Plot spatial point process residuals against a covariate

Usage

```
lurking(object, ...)

## S3 method for class 'ppm'
lurking(object, covariate,
                    type="eem",
                    cumulative=TRUE,
                    ...,
                    plot.it = TRUE,
                    plot.sd = is.poisson(object),
                    clipwindow=default.clipwindow(object),
                    rv = NULL,
```

```
                    envelope=FALSE, nsim=39, nrank=1,
                    typename,
                    covname,
                    oldstyle=FALSE,
                    check=TRUE,
                    verbose=TRUE,
                    nx=128,
                    splineargs=list(spar=0.5),
                    internal=NULL)

## S3 method for class 'ppp'
lurking(object, covariate,
                    type="eem",
                    cumulative=TRUE,
                    ...,
                    plot.it = TRUE,
                    plot.sd = is.poisson(object),
                    clipwindow=default.clipwindow(object),
                    rv = NULL,
                    envelope=FALSE, nsim=39, nrank=1,
                    typename,
                    covname,
                    oldstyle=FALSE,
                    check=TRUE,
                    verbose=TRUE,
                    nx=128,
                    splineargs=list(spar=0.5),
                    internal=NULL)
```

Arguments

- `object` The fitted point process model (an object of class "ppm") for which diagnostics should be produced. This object is usually obtained from `ppm`. Alternatively, `object` may be a point pattern (object of class "ppp").
- `covariate` The covariate against which residuals should be plotted. Either a numeric vector, a pixel image, or an `expression`. See *Details* below.
- `type` String indicating the type of residuals or weights to be computed. Choices include "eem", "raw", "inverse" and "pearson". See `diagnose.ppm` for all possible choices.
- `cumulative` Logical flag indicating whether to plot a cumulative sum of marks (`cumulative=TRUE`) or the derivative of this sum, a marginal density of the smoothed residual field (`cumulative=FALSE`).
- `...` Arguments passed to `plot.default` and `lines` to control the plot behaviour.
- `plot.it` Logical value indicating whether plots should be shown. If `plot.it=FALSE`, only the computed coordinates for the plots are returned. See *Value*.
- `plot.sd` Logical value indicating whether error bounds should be added to plot. The default is `TRUE` for Poisson models and `FALSE` for non-Poisson models. See Details.
- `clipwindow` If not `NULL` this argument indicates that residuals shall only be computed inside a subregion of the window containing the original point pattern data. Then

clipwindow should be a window object of class "owin".

rv
: Usually absent. If this argument is present, the point process residuals will not be calculated from the fitted model object, but will instead be taken directly from rv.

envelope
: Logical value indicating whether to compute simulation envelopes for the plot. Alternatively envelope may be a list of point patterns to use for computing the simulation envelopes, or an object of class "envelope" containing simulated point patterns.

nsim
: Number of simulated point patterns to be generated to produce the simulation envelope, if envelope=TRUE.

nrank
: Integer. Rank of the envelope value amongst the nsim simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.

typename
: Usually absent. If this argument is present, it should be a string, and will be used (in the axis labels of plots) to describe the type of residuals.

covname
: A string name for the covariate, to be used in axis labels of plots.

oldstyle
: Logical flag indicating whether error bounds should be plotted using the approximation given in the original paper (oldstyle=TRUE), or using the correct asymptotic formula (oldstyle=FALSE).

check
: Logical flag indicating whether the integrity of the data structure in object should be checked.

verbose
: Logical value indicating whether to print progress reports during Monte Carlo simulation.

nx
: Integer. Number of covariate values to be used in the plot.

splineargs
: A list of arguments passed to smooth.spline for the estimation of the derivatives in the case cumulative=FALSE.

internal
: Internal use only.

Details

This function generates a 'lurking variable' plot for a fitted point process model. Residuals from the model represented by object are plotted against the covariate specified by covariate. This plot can be used to reveal departures from the fitted model, in particular, to reveal that the point pattern depends on the covariate.

The function lurking is generic, with methods for ppm and ppp documented here, and possibly other methods.

The argument object would usually be a fitted point process model (object of class "ppm") produced by the model-fitting algorithm ppm). If object is a point pattern (object of class "ppp") then the model is taken to be the uniform Poisson process (Complete Spatial Randomness) fitted to this point pattern.

First the residuals from the fitted model (Baddeley et al, 2004) are computed at each quadrature point, or alternatively the 'exponential energy marks' (Stoyan and Grabarnik, 1991) are computed at each data point. The argument type selects the type of residual or weight. See diagnose.ppm for options and explanation.

A lurking variable plot for point processes (Baddeley et al, 2004) displays either the cumulative sum of residuals/weights (if cumulative = TRUE) or a kernel-weighted average of the residuals/weights (if cumulative = FALSE) plotted against the covariate. The empirical plot (solid lines) is shown together with its expected value assuming the model is true (dashed lines) and optionally also the pointwise two-standard-deviation limits (grey shading).

To be more precise, let $Z(u)$ denote the value of the covariate at a spatial location u.

- If `cumulative=TRUE` then we plot $H(z)$ against z, where $H(z)$ is the sum of the residuals over all quadrature points where the covariate takes a value less than or equal to z, or the sum of the exponential energy weights over all data points where the covariate takes a value less than or equal to z.
- If `cumulative=FALSE` then we plot $h(z)$ against z, where $h(z)$ is the derivative of $H(z)$, computed approximately by spline smoothing.

For the point process residuals $E(H(z)) = 0$, while for the exponential energy weights $E(H(z)) =$ area of the subset of the window satisfying $Z(u) <= z$.

If the empirical and theoretical curves deviate substantially from one another, the interpretation is that the fitted model does not correctly account for dependence on the covariate. The correct form (of the spatial trend part of the model) may be suggested by the shape of the plot.

If `plot.sd = TRUE`, then superimposed on the lurking variable plot are the pointwise two-standard-deviation error limits for $H(x)$ calculated for the inhomogeneous Poisson process. The default is `plot.sd = TRUE` for Poisson models and `plot.sd = FALSE` for non-Poisson models.

By default, the two-standard-deviation limits are calculated from the exact formula for the asymptotic variance of the residuals under the asymptotic normal approximation, equation (37) of Baddeley et al (2006). However, for compatibility with the original paper of Baddeley et al (2005), if `oldstyle=TRUE`, the two-standard-deviation limits are calculated using the innovation variance, an over-estimate of the true variance of the residuals.

The argument `covariate` is either a numeric vector, a pixel image, or an R language expression. If it is a numeric vector, it is assumed to contain the values of the covariate for each of the quadrature points in the fitted model. The quadrature points can be extracted by `quad.ppm(object)`.

If `covariate` is a pixel image, it is assumed to contain the values of the covariate at each location in the window. The values of this image at the quadrature points will be extracted.

Alternatively, if `covariate` is an `expression`, it will be evaluated in the same environment as the model formula used in fitting the model `object`. It must yield a vector of the same length as the number of quadrature points. The expression may contain the terms `x` and `y` representing the cartesian coordinates, and may also contain other variables that were available when the model was fitted. Certain variable names are reserved words; see `ppm`.

Note that lurking variable plots for the x and y coordinates are also generated by `diagnose.ppm`, amongst other types of diagnostic plots. This function is more general in that it enables the user to plot the residuals against any chosen covariate that may have been present.

For advanced use, even the values of the residuals/weights can be altered. If the argument `rv` is present, the residuals will not be calculated from the fitted model `object` but will instead be taken directly from the object `rv`. If `type = "eem"` then `rv` should be similar to the return value of `eem`, namely, a numeric vector with length equal to the number of data points in the original point pattern. Otherwise, `rv` should be similar to the return value of `residuals.ppm`, that is, `rv` should be an object of class `"msr"` (see `msr`) representing a signed measure.

Value

The (invisible) return value is an object belonging to the class `"lurk"`, for which there are methods for `plot` and `print`.

This object is a list containing two dataframes `empirical` and `theoretical`. The first dataframe `empirical` contains columns `covariate` and `value` giving the coordinates of the lurking variable plot. The second dataframe `theoretical` contains columns `covariate`, `mean` and `sd` giving the coordinates of the plot of the theoretical mean and standard deviation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

Baddeley, A., Møller, J. and Pakes, A.G. (2006) Properties of residuals for spatial point processes. *Annals of the Institute of Statistical Mathematics* **60**, 627–649.

Stoyan, D. and Grabarnik, P. (1991) Second-order characteristics for stochastic structures connected with Gibbs point processes. *Mathematische Nachrichten*, 151:95–100.

See Also

residuals.ppm, diagnose.ppm, residuals.ppm, qqplot.ppm, eem, ppm

Examples

```
lurking(nztrees, expression(x))
fit <- ppm(nztrees, ~x, Poisson())
lurking(fit, expression(x))
lurking(fit, expression(x), cumulative=FALSE)
```

---

lurking.mppm *Lurking Variable Plot for Multiple Point Patterns*

Description

Generate a lurking variable plot of spatial point process residuals against a covariate, for a model fitted to several point patterns.

Usage

```
## S3 method for class 'mppm'
lurking(object, covariate, type="eem",
                         ...,
                         separate = FALSE,
                         plot.it = TRUE,
                         covname, oldstyle = FALSE, nx = 512, main="")
```

Arguments

object	The fitted model. An object of class "mppm" representing a point process model fitted to several point patterns.
covariate	The covariate to be used on the horizontal axis. Either an expression which can be evaluated in the original data, or a list of pixel images, one image for each point pattern in the original data.
type	String indicating the type of residuals or weights to be computed. Choices include "eem", "raw", "inverse" and "pearson". See diagnose.ppm for all possible choices.

...	Additional arguments passed to lurking.ppm, including arguments controlling the plot.
separate	Logical value indicating whether to compute a separate lurking variable plot for each of the original point patterns. If FALSE (the default), a single lurking-variable plot is produced by combining residuals from all patterns.
plot.it	Logical value indicating whether plots should be shown. If plot.it=FALSE, only the computed coordinates for the plots are returned. See *Value*.
covname	A string name for the covariate, to be used in axis labels of plots.
oldstyle	Logical flag indicating whether error bounds should be plotted using the approximation given in the original paper (oldstyle=TRUE), or using the correct asymptotic formula (oldstyle=FALSE).
nx	Integer. Number of covariate values to be used in the plot.
main	Character string giving a main title for the plot.

Details

This function generates a 'lurking variable' plot for a point process model fitted to several point patterns. Residuals from the model represented by object are plotted against the covariate specified by covariate. This plot can be used to reveal departures from the fitted model.

The function lurking is generic. This is the method for the class mppm. The argument object must be a fitted point process model object of class "mppm") produced by the model-fitting algorithm mppm.

Value

If separate=FALSE (the default), the return value is an object belonging to the class "lurk", for which there are methods for plot and print. See lurking for details of the format.

If separate=TRUE, the result is a list of such objects, and also belongs to the class anylist so that it can be printed and plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, with thanks to Nicholas Read.

See Also

lurking.ppm

Examples

```
fit <- mppm(Points ~ Image + Group, demohyper)
lurking(fit, expression(Image), type="P")
lurking(fit, expression(Image), type="P", separate=TRUE)
```

lut *Lookup Tables*

Description

Create a lookup table.

Usage

```
lut(outputs, ..., range=NULL, breaks=NULL, inputs=NULL, gamma=1)
```

Arguments

outputs	Vector of output values
...	Ignored.
range	Interval of numbers to be mapped. A numeric vector of length 2, specifying the ends of the range of values to be mapped. Incompatible with `breaks` or `inputs`.
inputs	Input values to which the output values are associated. A factor or vector of the same length as `outputs`. Incompatible with `breaks` or `range`.
breaks	Breakpoints for the lookup table. A numeric vector of length equal to `length(outputs)+1`. Incompatible with `range` or `inputs`.
gamma	Exponent for gamma correction, when `range` is given. A single positive number. See Details.

Details

A lookup table is a function, mapping input values to output values.

The command `lut` creates an object representing a lookup table, which can then be used to control various behaviour in the **spatstat** package. It can also be used to compute the output value assigned to any input value.

The argument `outputs` specifies the output values to which input data values will be mapped. It should be a vector of any atomic type (e.g. numeric, logical, character, complex) or factor values.

Exactly one of the arguments `range`, `inputs` or `breaks` must be specified by name.

If `inputs` is given, then it should be a vector or factor, of the same length as `outputs`. The entries of `inputs` can be any atomic type (e.g. numeric, logical, character, complex) or factor values. The resulting lookup table associates the value `inputs[i]` with the value `outputs[i]`.

If `range` is given, then it determines the interval of the real number line that will be mapped. It should be a numeric vector of length 2. The interval will be divided evenly into bands, each of which is mapped to an entry of `outputs`. (If `gamma` is given, then the bands are equally spaced on a scale where the original values are raised to the power `gamma`.)

If `breaks` is given, then it determines intervals of the real number line which are mapped to each output value. It should be a numeric vector, of length at least 2, with entries that are in increasing order. Infinite values are allowed. Any number in the range between `breaks[i]` and `breaks[i+1]` will be mapped to the value `outputs[i]`.

The result is an object of class `"lut"`. There is a `print` method for this class. Some plot commands in the **spatstat** package accept an object of this class as a specification of a lookup table.

The result is also a function `f` which can be used to compute the output value assigned to any input data value. That is, `f(x)` returns the output value assigned to `x`. This also works for vectors of input data values.

Value

A function, which is also an object of class "lut".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

colourmap.

Examples

```
# lookup table for real numbers, using breakpoints
cr <- lut(factor(c("low", "medium", "high")), breaks=c(0,5,10,15))
cr
cr(3.2)
cr(c(3,5,7))
# lookup table for discrete set of values
ct <- lut(c(0,1), inputs=c(FALSE, TRUE))
ct(TRUE)
```

---

markconnect *Mark Connection Function*

Description

Estimate the marked connection function of a multitype point pattern.

Usage

```
markconnect(X, i, j, r=NULL,
         correction=c("isotropic", "Ripley", "translate"),
         method="density", ..., normalise=FALSE)
```

Arguments

X	The observed point pattern. An object of class "ppp" or something acceptable to as.ppp.
i	Number or character string identifying the type (mark value) of the points in X from which distances are measured.
j	Number or character string identifying the type (mark value) of the points in X to which distances are measured.
r	numeric vector. The values of the argument r at which the mark connection function $p_{ij}(r)$ should be evaluated. There is a sensible default.
correction	A character vector containing any selection of the options "isotropic", "Ripley" or "translate". It specifies the edge correction(s) to be applied.
method	A character vector indicating the user's choice of density estimation technique to be used. Options are "density", "loess", "sm" and "smrep".

`...` Arguments passed to the density estimation routine (`density`, `loess` or `sm.density`) selected by `method`.

`normalise` If `TRUE`, normalise the pair connection function by dividing it by $p_i p_j$, the estimated probability that randomly-selected points will have marks i and j.

Details

The mark connection function $p_{ij}(r)$ of a multitype point process X is a measure of the dependence between the types of two points of the process a distance r apart.

Informally $p_{ij}(r)$ is defined as the conditional probability, given that there is a point of the process at a location u and another point of the process at a location v separated by a distance $||u - v|| = r$, that the first point is of type i and the second point is of type j. See Stoyan and Stoyan (1994).

If the marks attached to the points of `X` are independent and identically distributed, then $p_{ij}(r) \equiv p_i p_j$ where p_i denotes the probability that a point is of type i. Values larger than this, $p_{ij}(r) > p_i p_j$, indicate positive association between the two types, while smaller values indicate negative association.

The argument `X` must be a point pattern (object of class `"ppp"`) or any data that are acceptable to `as.ppp`. It must be a multitype point pattern (a marked point pattern with factor-valued marks).

The argument `r` is the vector of values for the distance r at which $p_{ij}(r)$ is estimated. There is a sensible default.

This algorithm assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape.

Biases due to edge effects are treated in the same manner as in `Kest`. The edge corrections implemented here are

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is implemented only for rectangular and polygonal windows (not for binary masks).

translate Translation correction (Ohser, 1983). Implemented for all window geometries, but slow for complex windows.

Note that the estimator assumes the process is stationary (spatially homogeneous).

The mark connection function is estimated using density estimation techniques. The user can choose between

`"density"` which uses the standard kernel density estimation routine `density`, and works only for evenly-spaced `r` values;

`"loess"` which uses the function `loess` in the package **modreg**;

`"sm"` which uses the function `sm.density` in the package **sm** and is extremely slow;

`"smrep"` which uses the function `sm.density` in the package **sm** and is relatively fast, but may require manual control of the smoothing parameter `hmult`.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing numeric columns

`r` the values of the argument r at which the mark connection function $p_{ij}(r)$ has been estimated `theo` the theoretical value of $p_{ij}(r)$ when the marks attached to different points are independent

together with a column or columns named "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $p_{ij}(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

Multitype pair correlation pcfcross and multitype K-functions Kcross, Kdot.

Use alltypes to compute the mark connection functions between all pairs of types.

Mark correlation markcorr and mark variogram markvario for numeric-valued marks.

Examples

```
# Hughes' amacrine data
# Cells marked as 'on'/'off'
data(amacrine)
M <- markconnect(amacrine, "on", "off")
plot(M)

# Compute for all pairs of types at once
plot(alltypes(amacrine, markconnect))
```

---

markcorr *Mark Correlation Function*

Description

Estimate the marked correlation function of a marked point pattern.

Usage

```
markcorr(X, f = function(m1, m2) { m1 * m2}, r=NULL,
         correction=c("isotropic", "Ripley", "translate"),
         method="density", ..., weights=NULL,
         f1=NULL, normalise=TRUE, fargs=NULL)
```

Arguments

X	The observed point pattern. An object of class "ppp" or something acceptable to as.ppp.
f	Optional. Test function f used in the definition of the mark correlation function. An R function with at least two arguments. There is a sensible default.
r	Optional. Numeric vector. The values of the argument r at which the mark correlation function $k_f(r)$ should be evaluated. There is a sensible default.
correction	A character vector containing any selection of the options "isotropic", "Ripley", "translate", "translation", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
method	A character vector indicating the user's choice of density estimation technique to be used. Options are "density", "loess", "sm" and "smrep".
...	Arguments passed to the density estimation routine (density, loess or sm.density) selected by method.
weights	Optional numeric vector of weights for each data point in X.
f1	An alternative to f. If this argument is given, then f is assumed to take the form $f(u, v) = f_1(u)f_1(v)$.
normalise	If normalise=FALSE, compute only the numerator of the expression for the mark correlation.
fargs	Optional. A list of extra arguments to be passed to the function f or f1.

Details

By default, this command calculates an estimate of Stoyan's mark correlation $k_{mm}(r)$ for the point pattern.

Alternatively if the argument f or f1 is given, then it calculates Stoyan's generalised mark correlation $k_f(r)$ with test function f.

Theoretical definitions are as follows (see Stoyan and Stoyan (1994, p. 262)):

- For a point process X with numeric marks, Stoyan's mark correlation function $k_{mm}(r)$, is

$$k_{mm}(r) = \frac{E_{0u}[M(0)M(u)]}{E[M, M']}$$

  where E_{0u} denotes the conditional expectation given that there are points of the process at the locations 0 and u separated by a distance r, and where $M(0), M(u)$ denote the marks attached to these two points. On the denominator, M, M' are random marks drawn independently from the marginal distribution of marks, and E is the usual expectation.

- For a multitype point process X, the mark correlation is

$$k_{mm}(r) = \frac{P_{0u}[M(0)M(u)]}{P[M = M']}$$

  where P and P_{0u} denote the probability and conditional probability.

- The *generalised* mark correlation function $k_f(r)$ of a marked point process X, with test function f, is

$$k_f(r) = \frac{E_{0u}[f(M(0), M(u))]}{E[f(M, M')]}$$

The test function f is any function $f(m_1, m_2)$ with two arguments which are possible marks of the pattern, and which returns a nonnegative real value. Common choices of f are: for continuous nonnegative real-valued marks,

$$f(m_1, m_2) = m_1 m_2$$

for discrete marks (multitype point patterns),

$$f(m_1, m_2) = 1(m_1 = m_2)$$

and for marks taking values in $[0, 2\pi)$,

$$f(m_1, m_2) = \sin(m_1 - m_2)$$

.

Note that $k_f(r)$ is not a "correlation" in the usual statistical sense. It can take any nonnegative real value. The value 1 suggests "lack of correlation": if the marks attached to the points of `X` are independent and identically distributed, then $k_f(r) \equiv 1$. The interpretation of values larger or smaller than 1 depends on the choice of function f.

The argument `X` must be a point pattern (object of class `"ppp"`) or any data that are acceptable to `as.ppp`. It must be a marked point pattern.

The argument `f` determines the function to be applied to pairs of marks. It has a sensible default, which depends on the kind of marks in `X`. If the marks are numeric values, then `f <- function(m1, m2) { m1 * m2}` computes the product of two marks. If the marks are a factor (i.e. if `X` is a multitype point pattern) then `f <- function(m1, m2) { m1 == m2}` yields the value 1 when the two marks are equal, and 0 when they are unequal. These are the conventional definitions for numerical marks and multitype points respectively.

The argument `f` may be specified by the user. It must be an R function, accepting two arguments `m1` and `m2` which are vectors of equal length containing mark values (of the same type as the marks of `X`). (It may also take additional arguments, passed through `fargs`). It must return a vector of numeric values of the same length as `m1` and `m2`. The values must be non-negative, and `NA` values are not permitted.

Alternatively the user may specify the argument `f1` instead of `f`. This indicates that the test function f should take the form $f(u, v) = f_1(u)f_1(v)$ where $f_1(u)$ is given by the argument `f1`. The argument `f1` should be an R function with at least one argument. (It may also take additional arguments, passed through `fargs`).

The argument `r` is the vector of values for the distance r at which $k_f(r)$ is estimated.

This algorithm assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape.

Biases due to edge effects are treated in the same manner as in `Kest`. The edge corrections implemented here are

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is implemented only for rectangular and polygonal windows (not for binary masks).

translate Translation correction (Ohser, 1983). Implemented for all window geometries, but slow for complex windows.

Note that the estimator assumes the process is stationary (spatially homogeneous).

The numerator and denominator of the mark correlation function (in the expression above) are estimated using density estimation techniques. The user can choose between

`"density"` which uses the standard kernel density estimation routine `density`, and works only for evenly-spaced `r` values;

"loess" which uses the function loess in the package **modreg**;

"sm" which uses the function sm.density in the package **sm** and is extremely slow;

"smrep" which uses the function sm.density in the package **sm** and is relatively fast, but may require manual control of the smoothing parameter hmult.

If normalise=FALSE then the algorithm will compute only the numerator

$$c_f(r) = E_{0u}f(M(0), M(u))$$

of the expression for the mark correlation function.

Value

A function value table (object of class "fv") or a list of function value tables, one for each column of marks.

An object of class "fv" (see fv.object) is essentially a data frame containing numeric columns

r	the values of the argument r at which the mark correlation function $k_f(r)$ has been estimated
theo	the theoretical value of $k_f(r)$ when the marks attached to different points are independent, namely 1

together with a column or columns named "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the mark correlation function $k_f(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

Mark variogram markvario for numeric marks.

Mark connection function markconnect and multitype K-functions Kcross, Kdot for factor-valued marks.

Mark cross-correlation function markcrosscorr for point patterns with several columns of marks.

Kmark to estimate a cumulative function related to the mark correlation function.

Examples

```
# CONTINUOUS-VALUED MARKS:
# (1) Spruces
# marks represent tree diameter
# mark correlation function
ms <- markcorr(spruces)
plot(ms)

# (2) simulated data with independent marks
```

```
    X <- rpoispp(100)
    X <- X %mark% runif(npoints(X))
    ## Not run:
    Xc <- markcorr(X)
    plot(Xc)

## End(Not run)

    # MULTITYPE DATA:
    # Hughes' amacrine data
    # Cells marked as 'on'/'off'
    # (3) Kernel density estimate with Epanecnikov kernel
    # (as proposed by Stoyan & Stoyan)
    M <- markcorr(amacrine, function(m1,m2) {m1==m2},
                  correction="translate", method="density",
                  kernel="epanechnikov")
    plot(M)
    # Note: kernel="epanechnikov" comes from help(density)

    # (4) Same again with explicit control over bandwidth
    ## Not run:
    M <- markcorr(amacrine,
                  correction="translate", method="density",
                  kernel="epanechnikov", bw=0.02)
    # see help(density) for correct interpretation of 'bw'

## End(Not run)



   # weighted mark correlation
   Y <- subset(betacells, select=type)
   a <- marks(betacells)$area
   v <- markcorr(Y, weights=a)
```

---

markcrosscorr *Mark Cross-Correlation Function*

Description

Given a spatial point pattern with several columns of marks, this function computes the mark correlation function between each pair of columns of marks.

Usage

```
markcrosscorr(X, r = NULL,
              correction = c("isotropic", "Ripley", "translate"),
              method = "density", ..., normalise = TRUE, Xname = NULL)
```

Arguments

`X` The observed point pattern. An object of class "ppp" or something acceptable to `as.ppp`.

`r`	Optional. Numeric vector. The values of the argument r at which the mark correlation function $k_f(r)$ should be evaluated. There is a sensible default.
`correction`	A character vector containing any selection of the options `"isotropic"`, `"Ripley"`, `"translate"`, `"translation"`, `"none"` or `"best"`. It specifies the edge correction(s) to be applied. Alternatively `correction="all"` selects all options.
`method`	A character vector indicating the user's choice of density estimation technique to be used. Options are `"density"`, `"loess"`, `"sm"` and `"smrep"`.
`...`	Arguments passed to the density estimation routine (`density`, `loess` or `sm.density`) selected by `method`.
`normalise`	If `normalise=FALSE`, compute only the numerator of the expression for the mark correlation.
`Xname`	Optional character string name for the dataset `X`.

Details

First, all columns of marks are converted to numerical values. A factor with m possible levels is converted to m columns of dummy (indicator) values.

Next, each pair of columns is considered, and the mark cross-correlation is defined as

$$k_{mm}(r) = \frac{E_{0u}[M_i(0)M_j(u)]}{E[M_i, M_j]}$$

where E_{0u} denotes the conditional expectation given that there are points of the process at the locations 0 and u separated by a distance r. On the numerator, $M_i(0)$ and $M_j(u)$ are the marks attached to locations 0 and u respectively in the ith and jth columns of marks respectively. On the denominator, M_i and M_j are independent random values drawn from the ith and jth columns of marks, respectively, and E is the usual expectation.

Note that $k_{mm}(r)$ is not a "correlation" in the usual statistical sense. It can take any nonnegative real value. The value 1 suggests "lack of correlation": if the marks attached to the points of `X` are independent and identically distributed, then $k_{mm}(r) \equiv 1$.

The argument `X` must be a point pattern (object of class `"ppp"`) or any data that are acceptable to `as.ppp`. It must be a marked point pattern.

The cross-correlations are estimated in the same manner as for `markcorr`.

Value

A function array (object of class `"fasp"`) containing the mark cross-correlation functions for each possible pair of columns of marks.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

Rolf Turner `<r.turner@auckland.ac.nz>`

and Ege Rubak `<rubak@math.aau.dk>`

See Also

`markcorr`

Examples

```
# The dataset 'betacells' has two columns of marks:
#       'type' (factor)
#       'area' (numeric)
if(interactive()) plot(betacells)
plot(markcrosscorr(betacells))
```

---

marks *Marks of a Point Pattern*

Description

Extract or change the marks attached to a point pattern dataset.

Usage

```
marks(x, ...)

## S3 method for class 'ppp'
marks(x, ..., dfok=TRUE, drop=TRUE)

## S3 method for class 'ppx'
marks(x, ..., drop=TRUE)

marks(x, ...) <- value

## S3 replacement method for class 'ppp'
marks(x, ..., dfok=TRUE, drop=TRUE) <- value

## S3 replacement method for class 'ppx'
marks(x, ...) <- value

setmarks(x, value)

x %mark% value
```

Arguments

`x`	Point pattern dataset (object of class "ppp" or "ppx").
`...`	Ignored.
`dfok`	Logical. If `FALSE`, data frames of marks are not permitted and will generate an error.
`drop`	Logical. If `TRUE`, a data frame consisting of a single column of marks will be converted to a vector or factor.
`value`	Replacement value. A vector, data frame or hyperframe of mark values, or `NULL`.

Details

These functions extract or change the marks attached to the points of the point pattern `x`.

The expression `marks(x)` extracts the marks of `x`. The assignment `marks(x) <- value` assigns new marks to the dataset `x`, and updates the dataset `x` in the current environment. The expression `setmarks(x,value)` or equivalently `x %mark% value` returns a point pattern obtained by replacing the marks of `x` by `value`, but does not change the dataset `x` itself.

For point patterns in two-dimensional space (objects of class `"ppp"`) the marks can be a vector, a factor, or a data frame.

For general point patterns (objects of class "ppx") the marks can be a vector, a factor, a data frame or a hyperframe.

For the assignment `marks(x) <- value`, the `value` should be a vector or factor of length equal to the number of points in `x`, or a data frame or hyperframe with as many rows as there are points in `x`. If `value` is a single value, or a data frame or hyperframe with one row, then it will be replicated so that the same marks will be attached to each point.

To remove marks, use `marks(x) <- NULL` or `unmark(x)`.

Use `ppp` or `ppx` to create point patterns in more general situations.

Value

For `marks(x)`, the result is a vector, factor, data frame or hyperframe, containing the mark values attached to the points of `x`.

For `marks(x) <- value`, the result is the updated point pattern `x` (with the side-effect that the dataset `x` is updated in the current environment).

For `setmarks(x,value)` and `x %mark% value`, the return value is the point pattern obtained by replacing the marks of `x` by `value`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`ppp.object`, `ppx`, `unmark`, `hyperframe`

Examples

```
X <- amacrine
# extract marks
m <- marks(X)
# recode the mark values "off", "on" as 0, 1
marks(X) <- as.integer(m == "on")
```

marks.psp *Marks of a Line Segment Pattern*

Description

Extract or change the marks attached to a line segment pattern.

Usage

```
## S3 method for class 'psp'
marks(x, ..., dfok=TRUE)
## S3 replacement method for class 'psp'
marks(x, ...) <- value
```

Arguments

x	Line segment pattern dataset (object of class "psp").
...	Ignored.
dfok	Logical. If FALSE, data frames of marks are not permitted and will generate an error.
value	Vector or data frame of mark values, or NULL.

Details

These functions extract or change the marks attached to each of the line segments in the pattern x. They are methods for the generic functions marks and marks<- for the class "psp" of line segment patterns.

The expression marks(x) extracts the marks of x. The assignment marks(x) <- value assigns new marks to the dataset x, and updates the dataset x in the current environment.

The marks can be a vector, a factor, or a data frame.

For the assignment marks(x) <- value, the value should be a vector or factor of length equal to the number of segments in x, or a data frame with as many rows as there are segments in x. If value is a single value, or a data frame with one row, then it will be replicated so that the same marks will be attached to each segment.

To remove marks, use marks(x) <- NULL or unmark(x).

Value

For marks(x), the result is a vector, factor or data frame, containing the mark values attached to the line segments of x. If there are no marks, the result is NULL.

For marks(x) <- value, the result is the updated line segment pattern x (with the side-effect that the dataset x is updated in the current environment).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

psp.object, marks, marks<-

Examples

```
m <- data.frame(A=1:10, B=letters[1:10])
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin(), marks=m)

marks(X)
marks(X)[,2]
marks(X) <- 42
marks(X) <- NULL
```

---

marks.tess *Marks of a Tessellation*

Description

Extract or change the marks attached to the tiles of a tessellation.

Usage

```
## S3 method for class 'tess'
marks(x, ...)

## S3 replacement method for class 'tess'
marks(x, ...) <- value

## S3 method for class 'tess'
unmark(X)
```

Arguments

x,X	Tessellation (object of class "tess").
...	Ignored.
value	Vector or data frame of mark values, or NULL.

Details

These functions extract or change the marks attached to each of the tiles in the tessellation x. They are methods for the generic functions marks and marks<- for the class "tess" of tessellations.

The expression marks(x) extracts the marks of x. The assignment marks(x) <- value assigns new marks to the dataset x, and updates the dataset x in the current environment.

The marks can be a vector, a factor, or a data frame.

For the assignment marks(x) <- value, the value should be a vector or factor of length equal to the number of tiles in x, or a data frame with as many rows as there are tiles in x. If value is a single value, or a data frame with one row, then it will be replicated so that the same marks will be attached to each tile.

To remove marks, use marks(x) <- NULL or unmark(x).

Value

For `marks(x)`, the result is a vector, factor or data frame, containing the mark values attached to the tiles of `x`. If there are no marks, the result is `NULL`.

For `unmark(x)`, the result is the tessellation without marks.

For `marks(x) <- value`, the result is the updated tessellation `x` (with the side-effect that the dataset `x` is updated in the current environment).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`marks`, `marks<-`

Examples

```
D <- dirichlet(cells)
marks(D) <- tile.areas(D)
```

---

markstat — *Summarise Marks in Every Neighbourhood in a Point Pattern*

Description

Visit each point in a point pattern, find the neighbouring points, and summarise their marks

Usage

```
markstat(X, fun, N=NULL, R=NULL, ...)
```

Arguments

`X`	A marked point pattern. An object of class `"ppp"`.
`fun`	Function to be applied to the vector of marks.
`N`	Integer. If this argument is present, the neighbourhood of a point of `X` is defined to consist of the `N` points of `X` which are closest to it.
`R`	Nonnegative numeric value. If this argument is present, the neighbourhood of a point of `X` is defined to consist of all points of `X` which lie within a distance `R` of it.
`...`	extra arguments passed to the function `fun`. They must be given in the form `name=value`.

Details

This algorithm visits each point in the point pattern X, determines which points of X are "neighbours" of the current point, extracts the marks of these neighbouring points, applies the function fun to the marks, and collects the value or values returned by fun.

The definition of "neighbours" depends on the arguments N and R, exactly one of which must be given.

If N is given, then the neighbours of the current point are the N points of X which are closest to the current point (including the current point itself). If R is given, then the neighbourhood of the current point consists of all points of X which lie closer than a distance R from the current point.

Each point of X is visited; the neighbourhood of the current point is determined; the marks of these points are extracted as a vector v; then the function fun is called as:

fun(v, ...)

where ... are the arguments passed from the call to markstat.

The results of each call to fun are collected and returned according to the usual rules for apply and its relatives. See the section on **Value**.

This function is just a convenient wrapper for a common use of the function applynbd. For more complex tasks, use applynbd. To simply tabulate the marks in every R-neighbourhood, use marktable.

Value

Similar to the result of apply. if each call to fun returns a single numeric value, the result is a vector of dimension npoints(X), the number of points in X. If each call to fun returns a vector of the same length m, then the result is a matrix of dimensions c(m,n); note the transposition of the indices, as usual for the family of apply functions. If the calls to fun return vectors of different lengths, the result is a list of length npoints(X).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

applynbd, marktable, ppp.object, apply

Examples

```
trees <- longleaf


# average diameter of 5 closest neighbours of each tree
md <- markstat(trees, mean, N=5)

# range of diameters of trees within 10 metre radius
rd <- markstat(trees, range, R=10)
```

marktable *Tabulate Marks in Neighbourhood of Every Point in a Point Pattern*

Description

Visit each point in a point pattern, find the neighbouring points, and compile a frequency table of the marks of these neighbour points.

Usage

```
marktable(X, R, N, exclude=TRUE, collapse=FALSE)
```

Arguments

X	A marked point pattern. An object of class "ppp".
R	Neighbourhood radius. Incompatible with N.
N	Number of neighbours of each point. Incompatible with R.
exclude	Logical. If exclude=TRUE, the neighbours of a point do not include the point itself. If exclude=FALSE, a point belongs to its own neighbourhood.
collapse	Logical. If collapse=FALSE (the default) the results for each point are returned as separate rows of a table. If collapse=TRUE, the results are aggregated according to the type of point.

Details

This algorithm visits each point in the point pattern X, inspects all the neighbouring points within a radius R of the current point (or the N nearest neighbours of the current point), and compiles a frequency table of the marks attached to the neighbours.

The dataset X must be a multitype point pattern, that is, marks(X) must be a factor.

If collapse=FALSE (the default), the result is a two-dimensional contingency table with one row for each point in the pattern, and one column for each possible mark value. The [i,j] entry in the table gives the number of neighbours of point i that have mark j.

If collapse=TRUE, this contingency table is aggregated according to the type of point, so that the result is a contingency table with one row and one column for each possible mark value. The [i,j] entry in the table gives the number of neighbours of a point with mark i that have mark j.

To perform more complicated calculations on the neighbours of every point, use markstat or applynbd.

Value

A contingency table (object of class "table"). If collapse=FALSE, the table has one row for each point in X, and one column for each possible mark value. If collapse=TRUE, the table has one row and one column for each possible mark value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

markstat, applynbd, Kcross, ppp.object, table

Examples

```
head(marktable(amacrine, 0.1))
head(marktable(amacrine, 0.1, exclude=FALSE))
marktable(amacrine, N=1, collapse=TRUE)
```

---

markvario *Mark Variogram*

Description

Estimate the mark variogram of a marked point pattern.

Usage

```
markvario(X, correction = c("isotropic", "Ripley", "translate"),
r = NULL, method = "density", ..., normalise=FALSE)
```

Arguments

X	The observed point pattern. An object of class "ppp" or something acceptable to as.ppp. It must have marks which are numeric.
correction	A character vector containing any selection of the options "isotropic", "Ripley" or "translate". It specifies the edge correction(s) to be applied.
r	numeric vector. The values of the argument r at which the mark variogram $\gamma(r)$ should be evaluated. There is a sensible default.
method	A character vector indicating the user's choice of density estimation technique to be used. Options are "density", "loess", "sm" and "smrep".
...	Arguments passed to the density estimation routine (density, loess or sm.density) selected by method.
normalise	If TRUE, normalise the variogram by dividing it by the estimated mark variance.

Details

The mark variogram $\gamma(r)$ of a marked point process X is a measure of the dependence between the marks of two points of the process a distance r apart. It is informally defined as

$$\gamma(r) = E[\frac{1}{2}(M_1 - M_2)^2]$$

where $E[]$ denotes expectation and M_1, M_2 are the marks attached to two points of the process a distance r apart.

The mark variogram of a marked point process is analogous, but **not equivalent**, to the variogram of a random field in geostatistics. See Waelder and Stoyan (1996).

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the mark variogram $\gamma(r)$ has been estimated
theo	the theoretical value of $\gamma(r)$ when the marks attached to different points are independent; equal to the sample variance of the marks

together with a column or columns named "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $\gamma(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Cressie, N.A.C. (1991) *Statistics for spatial data*. John Wiley and Sons, 1991.

Mase, S. (1996) The threshold method for estimating annual rainfall. *Annals of the Institute of Statistical Mathematics* **48** (1996) 201-213.

Waelder, O. and Stoyan, D. (1996) On variograms in point process statistics. *Biometrical Journal* **38** (1996) 895-905.

See Also

Mark correlation function markcorr for numeric marks.

Mark connection function markconnect and multitype K-functions Kcross, Kdot for factor-valued marks.

Examples

```
# Longleaf Pine data
# marks represent tree diameter
data(longleaf)
# Subset of this large pattern
swcorner <- owin(c(0,100),c(0,100))
sub <- longleaf[ , swcorner]
# mark correlation function
mv <- markvario(sub)
plot(mv)
```

`matchingdist` *Distance for a Point Pattern Matching*

Description

Computes the distance associated with a matching between two point patterns.

Usage

```
matchingdist(matching, type = NULL, cutoff = NULL, q = NULL)
```

Arguments

`matching`	A point pattern matching (an object of class `"pppmatching"`).
`type`	A character string giving the type of distance to be computed. One of `"spa"`, `"ace"` or `"mat"`. See details below.
`cutoff`	The value > 0 at which interpoint distances are cut off.
`q`	The order of the average that is applied to the interpoint distances. May be `Inf`, in which case the maximum of the interpoint distances is taken.

Details

Computes the distance specified by `type`, `cutoff`, and `order` for a point matching. If any of these arguments are not provided, the function uses the corresponding elements of `matching` (if available).

For the type `"spa"` (subpattern assignment) it is assumed that the points of the point pattern with the smaller cardinality m are matched to a m-point subpattern of the point pattern with the larger cardinality n in a 1-1 way. The distance is then given as the q-th order average of the m distances between matched points (minimum of Euclidean distance and `cutoff`) and $n-m$ "penalty distances" of value `cutoff`.

For the type `"ace"` (assignment only if cardinalities equal) the matching is assumed to be 1-1 if the cardinalities of the point patterns are the same, in which case the q-th order average of the matching distances (minimum of Euclidean distance and `cutoff`) is taken. If the cardinalities are different, the matching may be arbitrary and the distance returned is always equal to `cutoff`.

For the type `mat` (mass transfer) it is assumed that each point of the point pattern with the smaller cardinality m has mass 1, each point of the point pattern with the larger cardinality n has mass m/n, and fractions of these masses are matched in such a way that each point contributes exactly its mass. The distance is then given as the q-th order weighted average of all distances (minimum of Euclidean distance and `cutoff`) of (partially) matched points with weights equal to the fractional masses divided by m.

If the cardinalities of the two point patterns are equal, `matchingdist(m, type, cutoff, q)` yields the same result no matter if `type` is `"spa"`, `"ace"` or `"mat"`.

Value

Numeric value of the distance associated with the matching.

Author(s)

Dominic Schuhmacher <dominic.schuhmacher@stat.unibe.ch> http://www.dominic.schuhmacher.name

See Also

pppdist pppmatching.object

Examples

```
   # an optimal matching
   X <- runifpoint(20)
   Y <- runifpoint(20)
   m.opt <- pppdist(X, Y)
   summary(m.opt)
   matchingdist(m.opt)
        # is the same as the distance given by summary(m.opt)

   # sequential nearest neighbour matching
   # (go through all points of point pattern X in sequence
   # and match each point with the closest point of Y that is
   # still unmatched)
   am <- matrix(0, 20, 20)
   h <- matrix(c(1:20, rep(0,20)), 20, 2)
   h[1,2] = nncross(X[1],Y)[1,2]
   for (i in 2:20) {
     nn <- nncross(X[i],Y[-h[1:(i-1),2]])[1,2]
     h[i,2] <- ((1:20)[-h[1:(i-1),2]])[nn]
   }
   am[h] <- 1
   m.nn <- pppmatching(X, Y, am)
   matchingdist(m.nn, type="spa", cutoff=1, q=1)
        # is >= the distance obtained for m.opt
        # in most cases strictly >

   ## Not run:
     par(mfrow=c(1,2))
     plot(m.opt)
     plot(m.nn)
     text(X$x, X$y, 1:20, pos=1, offset=0.3, cex=0.8)

## End(Not run)
```

---

matclust.estK — *Fit the Matern Cluster Point Process by Minimum Contrast*

Description

Fits the Matern Cluster point process to a point pattern dataset by the Method of Minimum Contrast.

Usage

```
matclust.estK(X, startpar=c(kappa=1,scale=1), lambda=NULL,
             q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...)
```

Arguments

X	Data to which the Matern Cluster model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the Matern Cluster process.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.

Details

This algorithm fits the Matern Cluster point process model to a point pattern dataset by the Method of Minimum Contrast, using the K function.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The K function of the point pattern will be computed using Kest, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the K function, and this object should have been obtained by a call to Kest or one of its relatives.

The algorithm fits the Matern Cluster point process to X, by finding the parameters of the Matern Cluster model which give the closest match between the theoretical K function of the Matern Cluster process and the observed K function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The Matern Cluster point process is described in Møller and Waagepetersen (2003, p. 62). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent are independent and uniformly distributed inside a circle of radius R centred on the parent point, where R is equal to the parameter scale. The named vector of stating values can use either R or scale as the name of the second component, but the latter is recommended for consistency with other cluster models.

The theoretical K-function of the Matern Cluster process is

$$K(r) = \pi r^2 + \frac{1}{\kappa} h(\frac{r}{2R})$$

where the radius R is the parameter scale and

$$h(z) = 2 + \frac{1}{\pi}[(8z^2 - 4)\text{arccos}(z) - 2\text{arcsin}(z) + 4z\sqrt{(1-z^2)^3} - 6z\sqrt{1-z^2}]$$

for $z <= 1$, and $h(z) = 1$ for $z > 1$. The theoretical intensity of the Matern Cluster process is $\lambda = \kappa\mu$.

In this algorithm, the Method of Minimum Contrast is first used to find optimal values of the parameters κ and R. Then the remaining parameter μ is inferred from the estimated intensity λ.

If the argument lambda is provided, then this is used as the value of λ. Otherwise, if X is a point pattern, then λ will be estimated from X. If X is a summary statistic and lambda is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as NA.

The remaining arguments `rmin,rmax,q,p` control the method of minimum contrast; see `mincontrast`.

The Matern Cluster process can be simulated, using `rMatClust`.

Homogeneous or inhomogeneous Matern Cluster models can also be fitted using the function `kppm`.

The optimisation algorithm can be controlled through the additional arguments `"..."` which are passed to the optimisation function `optim`. For example, to constrain the parameter values to a certain range, use the argument `method="L-BFGS-B"` to select an optimisation algorithm that respects box constraints, and use the arguments `lower` and `upper` to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class `"minconfit"`. There are methods for printing and plotting this object. It contains the following main components:

`par`	Vector of fitted parameter values.
`fit`	Function value table (object of class `"fv"`) containing the observed values of the summary statistic (`observed`) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Rasmus Waagepetersen `<rw@math.auc.dk>` Adapted for **spatstat** by Adrian Baddeley `<Adrian.Baddeley@curtin.edu`

References

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

`kppm`, `lgcp.estK`, `thomas.estK`, `mincontrast`, `Kest`, `rMatClust` to simulate the fitted model.

Examples

```
data(redwood)
u <- matclust.estK(redwood, c(kappa=10, scale=0.1))
u
plot(u)
```

`matclust.estpcf` *Fit the Matern Cluster Point Process by Minimum Contrast Using Pair Correlation*

Description

Fits the Matern Cluster point process to a point pattern dataset by the Method of Minimum Contrast using the pair correlation function.

Usage

```
matclust.estpcf(X, startpar=c(kappa=1,scale=1), lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...,
            pcfargs=list())
```

Arguments

X	Data to which the Matern Cluster model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the Matern Cluster process.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.
pcfargs	Optional list containing arguments passed to pcf.ppp to control the smoothing in the estimation of the pair correlation function.

Details

This algorithm fits the Matern Cluster point process model to a point pattern dataset by the Method of Minimum Contrast, using the pair correlation function.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The pair correlation function of the point pattern will be computed using pcf, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the pair correlation function, and this object should have been obtained by a call to pcf or one of its relatives.

The algorithm fits the Matern Cluster point process to X, by finding the parameters of the Matern Cluster model which give the closest match between the theoretical pair correlation function of the Matern Cluster process and the observed pair correlation function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The Matern Cluster point process is described in Møller and Waagepetersen (2003, p. 62). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent are independent and uniformly distributed inside a circle of radius R centred on the parent point, where R is equal to the parameter scale. The named vector of stating values can use either R or scale as the name of the second component, but the latter is recommended for consistency with other cluster models.

The theoretical pair correlation function of the Matern Cluster process is

$$g(r) = 1 + \frac{1}{4\pi R \kappa r} h(\frac{r}{2R})$$

where the radius R is the parameter scale and

$$h(z) = \frac{16}{\pi}[z \arccos(z) - z^2\sqrt{1-z^2}]$$

for $z <= 1$, and $h(z) = 0$ for $z > 1$. The theoretical intensity of the Matern Cluster process is $\lambda = \kappa\mu$.

In this algorithm, the Method of Minimum Contrast is first used to find optimal values of the parameters κ and R. Then the remaining parameter μ is inferred from the estimated intensity λ.

If the argument `lambda` is provided, then this is used as the value of λ. Otherwise, if `X` is a point pattern, then λ will be estimated from `X`. If `X` is a summary statistic and `lambda` is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as `NA`.

The remaining arguments `rmin,rmax,q,p` control the method of minimum contrast; see `mincontrast`.

The Matern Cluster process can be simulated, using `rMatClust`.

Homogeneous or inhomogeneous Matern Cluster models can also be fitted using the function `kppm`.

The optimisation algorithm can be controlled through the additional arguments `"..."` which are passed to the optimisation function `optim`. For example, to constrain the parameter values to a certain range, use the argument `method="L-BFGS-B"` to select an optimisation algorithm that respects box constraints, and use the arguments `lower` and `upper` to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class `"minconfit"`. There are methods for printing and plotting this object. It contains the following main components:

`par`	Vector of fitted parameter values.
`fit`	Function value table (object of class `"fv"`) containing the observed values of the summary statistic (`observed`) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

`kppm`, `matclust.estK`, `thomas.estpcf`, `thomas.estK`, `lgcp.estK`, `mincontrast`, `pcf`, `rMatClust` to simulate the fitted model.

Examples

```
data(redwood)
u <- matclust.estpcf(redwood, c(kappa=10, R=0.1))
u
plot(u, legendpos="topright")
```

Math.im	*S3 Group Generic methods for images*

Description

These are group generic methods for images of class "im", which allows for usual mathematical functions and operators to be applied directly to images. See Details for a list of implemented functions.

Usage

```
## S3 methods for group generics have prototypes:
Math(x, ...)
Ops(e1, e2)
Complex(z)
Summary(..., na.rm=FALSE, drop=TRUE)
```

Arguments

x, z, e1, e2	objects of class "im".
...	further arguments passed to methods.
na.rm,drop	Logical values specifying whether missing values should be removed. This will happen if either na.rm=TRUE or drop=TRUE. See Details.

Details

Below is a list of mathematical functions and operators which are defined for images. Not all functions will make sense for all types of images. For example, none of the functions in the "Math" group make sense for character-valued images. Note that the "Ops" group methods are implemented using eval.im, which tries to harmonise images via harmonise.im if they aren't compatible to begin with.

1. Group "Math":
   - abs, sign, sqrt,
     floor, ceiling, trunc,
     round, signif
   - exp, log, expm1, log1p,
     cos, sin, tan,
     cospi, sinpi, tanpi,
     acos, asin, atan
     cosh, sinh, tanh,
     acosh, asinh, atanh
   - lgamma, gamma, digamma, trigamma
   - cumsum, cumprod, cummax, cummin
2. Group "Ops":

- `"+"`, `"-"`, `"*"`, `"/"`, `"^"`, `"%%"`, `"%/%"`
- `"&"`, `"|"`, `"!"`
- `"=="`, `"!="`, `"<"`, `"<="`, `">="`, `">"`

3. Group `"Summary"`:
   - `all`, `any`
   - `sum`, `prod`
   - `min`, `max`
   - `range`
4. Group `"Complex"`:
   - `Arg`, `Conj`, `Im`, `Mod`, `Re`

For the `Summary` group, the generic has an argument `na.rm=FALSE`, but for pixel images it makes sense to set `na.rm=TRUE` so that pixels outside the domain of the image are ignored. To enable this, we added the argument `drop`. Pixel values that are `NA` are removed if `drop=TRUE` or if `na.rm=TRUE`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> and Kassel Hingee.

See Also

`eval.im` for evaluating expressions involving images.

Examples

```
## Convert gradient values to angle of inclination:
V <- atan(bei.extra$grad) * 180/pi
## Make logical image which is TRUE when heat equals 'Moderate':
A <- (gorillas.extra$heat == "Moderate")
## Summary:
any(A)
## Complex:
Z <- exp(1 + V * 1i)
Z
Re(Z)
```

Math.imlist — *S3 Group Generic methods for List of Images*

Description

These are group generic methods for the class `"imlist"` of lists of images. These methods allows the usual mathematical functions and operators to be applied directly to lists of images. See Details for a list of implemented functions.

Usage

```
## S3 methods for group generics have prototypes:
Math(x, ...)
Ops(e1, e2)
Complex(z)
Summary(..., na.rm = TRUE)
```

Arguments

`x, z, e1, e2`	objects of class `"imlist"`.
`...`	further arguments passed to methods.
`na.rm`	logical: should missing values be removed?

Details

Below is a list of mathematical functions and operators which are defined for lists of images. Not all functions will make sense for all types of images. For example, none of the functions in the `"Math"` group make sense for character-valued images. Note that the `"Ops"` group methods are implemented using `eval.im`, which tries to harmonise images via `harmonise.im` if they aren't compatible to begin with.

1. Group `"Math"`:
   - `abs`, `sign`, `sqrt`,
     `floor`, `ceiling`, `trunc`,
     `round`, `signif`
   - `exp`, `log`, `expm1`, `log1p`,
     `cos`, `sin`, `tan`,
     `cospi`, `sinpi`, `tanpi`,
     `acos`, `asin`, `atan`
     `cosh`, `sinh`, `tanh`,
     `acosh`, `asinh`, `atanh`
   - `lgamma`, `gamma`, `digamma`, `trigamma`
   - `cumsum`, `cumprod`, `cummax`, `cummin`
2. Group `"Ops"`:
   - `"+"`, `"-"`, `"*"`, `"/"`, `"^"`, `"%%"`, `"%/%"`
   - `"&"`, `"|"`, `"!"`
   - `"=="`, `"!="`, `"<"`, `"<="`, `">="`, `">"`
3. Group `"Summary"`:
   - `all`, `any`
   - `sum`, `prod`
   - `min`, `max`
   - `range`
4. Group `"Complex"`:
   - `Arg`, `Conj`, `Im`, `Mod`, `Re`

Value

The result of "Math", "Ops" and "Complex" group operations is another list of images. The result of "Summary" group operations is a numeric vector of length 1 or 2.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

Math.im or eval.im for evaluating expressions involving images.

Examples

```
a <- Smooth(finpines, 2)
log(a)/2 - sqrt(a)
range(a)
```

Math.linim *S3 Group Generic Methods for Images on a Linear Network*

Description

These are group generic methods for images of class "linim", which allows for usual mathematical functions and operators to be applied directly to pixel images on a linear network. See Details for a list of implemented functions.

Usage

```
## S3 methods for group generics have prototypes:
Math(x, ...)
Ops(e1, e2)
Complex(z)
Summary(..., na.rm = FALSE)
```

Arguments

x, z, e1, e2	objects of class "linim".
...	further arguments passed to methods.
na.rm	logical: should missing values be removed?

Details

An object of class "linim" represents a pixel image on a linear network. See linim.

Below is a list of mathematical functions and operators which are defined for these images. Not all functions will make sense for all types of images. For example, none of the functions in the "Math" group make sense for character-valued images. Note that the "Ops" group methods are implemented using eval.linim.

1. Group "Math":
   - abs, sign, sqrt,
     floor, ceiling, trunc,
     round, signif
   - exp, log, expm1, log1p,
     cos, sin, tan,
     cospi, sinpi, tanpi,
     acos, asin, atan
     cosh, sinh, tanh,
     acosh, asinh, atanh
   - lgamma, gamma, digamma, trigamma
   - cumsum, cumprod, cummax, cummin
2. Group "Ops":
   - "+", "-", "*", "/", "^", "%%", "%/%"
   - "&", "|", "!"
   - "==", "!=", "<", "<=", ">=", ">"
3. Group "Summary":
   - all, any
   - sum, prod
   - min, max
   - range
4. Group "Complex":
   - Arg, Conj, Im, Mod, Re

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

eval.linim for evaluating expressions involving images.

Examples

```
fx <- function(x,y,seg,tp) { (x - y)^2 }
fL <- linfun(fx, simplenet)
Z <- as.linim(fL)
A <- Z+2
A <- -Z
A <- sqrt(Z)
A <- !(Z > 0.1)
```

matrixpower *Power of a Matrix*

Description

Evaluate a specified power of a matrix.

Usage

```
matrixpower(x, power, complexOK = TRUE)
matrixsqrt(x, complexOK = TRUE)
matrixinvsqrt(x, complexOK = TRUE)
```

Arguments

x	A square matrix containing numeric or complex values.
power	A numeric value giving the power (exponent) to which x should be raised.
complexOK	Logical value indicating whether the result is allowed to be complex.

Details

These functions raise the matrix x to the desired power: matrixsqrt takes the square root, matrixinvsqrt takes the inverse square root, and matrixpower takes the specified power of x.

Up to numerical error, matrixpower(x, 2) should be equivalent to x %*% x, and matrixpower(x, -1) should be equivalent to solve(x), the inverse of x.

The square root y <- matrixsqrt(x) should satisfy y %*% y = x. The inverse square root z <- matrixinvsqrt(x) should satisfy z %*% z = solve(x).

Computations are performed using the eigen decomposition (eigen).

Value

A matrix of the same size as x containing numeric or complex values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

eigen, svd

Examples

```
x <- matrix(c(10,2,2,1), 2, 2)
y <- matrixsqrt(x)
y
y %*% y
z <- matrixinvsqrt(x)
z %*% y
matrixpower(x, 0.1)
```

maxnndist *Compute Minimum or Maximum Nearest-Neighbour Distance*

Description

A faster way to compute the minimum or maximum nearest-neighbour distance in a point pattern.

Usage

```
minnndist(X, positive=FALSE)
maxnndist(X, positive=FALSE)
```

Arguments

X	A point pattern (object of class "ppp").
positive	Logical. If FALSE (the default), compute the usual nearest-neighbour distance. If TRUE, ignore coincident points, so that the nearest neighbour distance for each point is greater than zero.

Details

These functions find the minimum and maximum values of nearest-neighbour distances in the point pattern X. minnndist(X) and maxnndist(X) are equivalent to, but faster than, min(nndist(X)) and max(nndist(X)) respectively.

The value is NA if npoints(X) < 2.

Value

A single numeric value (possibly NA).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

nndist

Examples

```
min(nndist(swedishpines))
minnndist(swedishpines)

max(nndist(swedishpines))
maxnndist(swedishpines)

minnndist(lansing, positive=TRUE)

if(interactive()) {
   X <- rpoispp(1e6)
   system.time(min(nndist(X)))
```

```
    system.time(minnndist(X))
  }
```

mean.im *Mean and Median of Pixel Values in an Image*

Description

Calculates the mean or median of the pixel values in a pixel image.

Usage

```
## S3 method for class 'im'
## mean(x, trim=0, na.rm=TRUE, ...)

## S3 method for class 'im'
## median(x, na.rm=TRUE)        [R < 3.4.0]
## median(x, na.rm=TRUE, ...)   [R >= 3.4.0]
```

Arguments

x	A pixel image (object of class "im").
na.rm	Logical value indicating whether NA values should be stripped before the computation proceeds.
trim	The fraction (0 to 0.5) of pixel values to be trimmed from each end of their range, before the mean is computed.
...	Ignored.

Details

These functions calculate the mean and median of the pixel values in the image x.

An object of class "im" describes a pixel image. See im.object) for details of this class.

The function mean.im is a method for the generic function mean for the class "im". Similarly median.im is a method for the generic median.

If the image x is logical-valued, the mean value of x is the fraction of pixels that have the value TRUE. The median is not defined.

If the image x is factor-valued, then the mean of x is the mean of the integer codes of the pixel values. The median is are not defined.

Other mathematical operations on images are supported by Math.im, Summary.im and Complex.im.

Other information about an image can be obtained using summary.im or quantile.im.

Value

A single number.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> and Kassel Hingee.

See Also

Math.im for other operations.

Generics and default methods: mean, median.

quantile.im, anyNA.im, im.object, summary.im.

Examples

```
X <- as.im(function(x,y) {x^2}, unit.square())
mean(X)
median(X)
mean(X, trim=0.05)
```

---

mean.linim *Mean, Median, Quantiles of Pixel Values on a Linear Network*

Description

Calculates the mean, median, or quantiles of the pixel values in a pixel image on a linear network.

Usage

```
  ## S3 method for class 'linim'
mean(x, ...)

  ## S3 method for class 'linim'
median(x, ...)

  ## S3 method for class 'linim'
quantile(x, probs=seq(0,1,0.25), ...)
```

Arguments

x	A pixel image on a linear network (object of class "linim").
probs	Vector of probabilities for which quantiles should be calculated.
...	Arguments passed to other methods.

Details

These functions calculate the mean, median and quantiles of the pixel values in the image x on a linear network.

An object of class "linim" describes a pixel image on a linear network. See linim.

The functions described here are methods for the generic mean, median and quantile for the class "linim".

Value

For mean and median, a single number. For quantile, a numeric vector of the same length as probs.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

mean, median, quantile,

mean.im.

Examples

```
M <- as.mask.psp(as.psp(simplenet))
Z <- as.im(function(x,y) {x-y}, W=M)
X <- linim(simplenet, Z)
X
mean(X)
median(X)
quantile(X)
```

---

measureVariation *Positive and Negative Parts, and Variation, of a Measure*

Description

Given a measure A (object of class "msr") these functions find the positive part, negative part and variation of A.

Usage

```
measurePositive(x)
measureNegative(x)
measureVariation(x)
totalVariation(x)
```

Arguments

x	A measure (object of class "msr").

Details

The functions measurePositive and measureNegative return the positive and negative parts of the measure, and measureVariation returns the variation (sum of positive and negative parts). The function totalVariation returns the total variation norm.

If μ is a signed measure, it can be represented as

$$\mu = \mu_+ - \mu_-$$

where μ_+ and μ_- are *nonnegative* measures called the positive and negative parts of μ. In a nutshell, the positive part of μ consists of all positive contributions or increments, and the negative part consists of all negative contributions multiplied by -1.

The variation $|\mu|$ is defined by

$$\mu = \mu_+ + \mu_-$$

and is also a nonnegative measure.

The total variation norm is the integral of the variation.

Value

The result of measurePositive, measureNegative and measureVariation is another measure (object of class "msr") on the same spatial domain. The result of totalVariation is a non-negative number.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Halmos, P.R. (1950) *Measure Theory*. Van Nostrand.

See Also

msr, with.msr, split.msr

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
rp <- residuals(fit, type="pearson")

measurePositive(rp)
measureNegative(rp)
measureVariation(rp)

# total variation norm
totalVariation(rp)
```

---

mergeLevels *Merge Levels of a Factor*

Description

Specified levels of the factor will be merged into a single level.

Usage

```
mergeLevels(.f, ...)
```

Arguments

.f	A factor (or a factor-valued pixel image or a point pattern with factor-valued marks).
...	List of name=value pairs, where name is the new merged level, and value is the vector of old levels that will be merged.

Details

This utility function takes a factor `.f` and merges specified levels of the factor.

The grouping is specified by the arguments `...` which must each be given in the form `new=old`, where `new` is the name for the new merged level, and `old` is a character vector containing the old levels that are to be merged.

The result is a new factor (or factor-valued object), in which the levels listed in `old` have been replaced by a single level `new`.

An argument of the form `name=character(0)` or `name=NULL` is interpreted to mean that all other levels of the old factor should be mapped to `name`.

Value

Another factor of the same length as `.f` (or object of the same kind as `.f`).

Tips for manipulating factor levels

To remove unused levels from a factor `f`, just type `f <- factor(f)`.

To change the ordering of levels in a factor, use `factor(f, levels=l)` or `relevel(f, ref)`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`factor`, `relevel`

Examples

```
likert <- c("Strongly Agree", "Agree", "Neutral",
            "Disagree", "Strongly Disagree")
answers <- factor(sample(likert, 15, replace=TRUE), levels=likert)
answers
mergeLevels(answers, Positive=c("Strongly Agree", "Agree"),
                     Negative=c("Strongly Disagree", "Disagree"))
```

`methods.box3` *Methods for Three-Dimensional Box*

Description

Methods for class `"box3"`.

Usage

```
  ## S3 method for class 'box3'
print(x, ...)
  ## S3 method for class 'box3'
unitname(x)
  ## S3 replacement method for class 'box3'
unitname(x) <- value
```

Arguments

x	Object of class "box3" representing a three-dimensional box.
...	Other arguments passed to print.default.
value	Name of the unit of length. See unitname.

Details

These are methods for the generic functions print and unitname for the class "box3" of three-dimensional boxes.

The print method prints a description of the box, while the unitname method extracts the name of the unit of length in which the box coordinates are expressed.

Value

For print.box3 the value is NULL. For unitname.box3 an object of class "units".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

box3, print, unitname

Examples

```
X <- box3(c(0,10),c(0,10),c(0,5), unitname=c("metre", "metres"))
X
unitname(X)
# Northern European usage
unitname(X) <- "meter"
```

methods.boxx *Methods for Multi-Dimensional Box*

Description

Methods for class "boxx".

Usage

```
  ## S3 method for class 'boxx'
print(x, ...)
  ## S3 method for class 'boxx'
unitname(x)
  ## S3 replacement method for class 'boxx'
unitname(x) <- value
```

Arguments

x	Object of class "boxx" representing a multi-dimensional box.
...	Other arguments passed to print.default.
value	Name of the unit of length. See unitname.

Details

These are methods for the generic functions print and unitname for the class "boxx" of multi-dimensional boxes.

The print method prints a description of the box, while the unitname method extracts the name of the unit of length in which the box coordinates are expressed.

Value

For print.boxx the value is NULL. For unitname.boxx an object of class "units".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

boxx, print, unitname

Examples

```
X <- boxx(c(0,10),c(0,10),c(0,5),c(0,1), unitname=c("metre", "metres"))
X
unitname(X)
# Northern European usage
unitname(X) <- "meter"
```

methods.dppm *Methods for Determinantal Point Process Models*

Description

These are methods for the class "dppm".

Usage

```
## S3 method for class 'dppm'
coef(object, ...)
## S3 method for class 'dppm'
formula(x, ...)
## S3 method for class 'dppm'
print(x, ...)
## S3 method for class 'dppm'
terms(x, ...)
## S3 method for class 'dppm'
labels(object, ...)
```

Arguments

x,object	An object of class "dppm", representing a fitted determinantal point process model.
...	Arguments passed to other methods.

Details

These functions are methods for the generic commands coef, formula, print, terms and labels for the class "dppm".

An object of class "dppm" represents a fitted determinantal point process model. It is obtained from dppm.

The method coef.dppm returns the vector of *regression coefficients* of the fitted model. It does not return the interaction parameters.

Value

See the help files for the corresponding generic functions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

dppm, plot.dppm, predict.dppm, simulate.dppm, as.ppm.dppm.

Examples

```
fit <- dppm(swedishpines ~ x + y, dppGauss())
coef(fit)
formula(fit)
tf <- terms(fit)
labels(fit)
```

---

methods.fii *Methods for Fitted Interactions*

Description

These are methods specifically for the class "fii" of fitted interpoint interactions.

Usage

```
## S3 method for class 'fii'
print(x, ...)

## S3 method for class 'fii'
coef(object, ...)

## S3 method for class 'fii'
plot(x, ...)

## S3 method for class 'fii'
summary(object,...)

## S3 method for class 'summary.fii'
print(x, ...)

## S3 method for class 'summary.fii'
coef(object, ...)
```

Arguments

x,object	An object of class "fii" representing a fitted interpoint interaction.
...	Arguments passed to other methods.

Details

These are methods for the class "fii". An object of class "fii" represents a fitted interpoint interaction. It is usually obtained by using the command fitin to extract the fitted interaction part of a fitted point process model. See fitin for further explanation of this class.

The commands listed here are methods for the generic functions print, summary, plot and coef for objects of the class "fii".

Following the usual convention, summary.fii returns an object of class summary.fii, for which there is a print method. The effect is that, when the user types summary(x), the summary is printed, but when the user types y <- summary(x), the summary information is saved.

The method `coef.fii` extracts the canonical coefficients of the fitted interaction, and returns them as a numeric vector. The method `coef.summary.fii` transforms these values into quantities that are more easily interpretable, in a format that depends on the particular model.

There are also methods for the generic commands `reach` and `as.interact`, described elsewhere.

Value

The `print` and `plot` methods return `NULL`.

The `summary` method returns an object of class `summary.fii`.

`coef.fii` returns a numeric vector. `coef.summary.fii` returns data whose structure depends on the model.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

See Also

`fitin`, `reach.fii`, `as.interact.fii`

Examples

```
mod <- ppm(cells, ~1, Strauss(0.1))
f <- fitin(mod)
f
summary(f)
plot(f)
coef(f)
coef(summary(f))
```

---

methods.funxy *Methods for Spatial Functions*

Description

Methods for objects of the class `"funxy"`.

Usage

```
## S3 method for class 'funxy'
contour(x, ...)
## S3 method for class 'funxy'
persp(x, ...)
## S3 method for class 'funxy'
plot(x, ...)
```

Arguments

`x`	Object of class `"funxy"` representing a function of x, y coordinates.
`...`	Named arguments controlling the plot. See Details.

Details

These are methods for the generic functions `plot`, `contour` and `persp` for the class `"funxy"` of spatial functions.

Objects of class `"funxy"` are created, for example, by the commands `distfun` and `funxy`.

The `plot`, `contour` and `persp` methods first convert `x` to a pixel image object using `as.im`, then display it using `plot.im`, `contour.im` or `persp.im`.

Additional arguments `...` are either passed to `as.im.function` to control the spatial resolution of the pixel image, or passed to `contour.im`, `persp.im` or `plot.im` to control the appearance of the plot.

Value

`NULL`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`funxy`, `distfun`, `as.im`, `plot.im`, `persp.im`, `contour.im`, `spatstat.options`

Examples

```
data(letterR)
f <- distfun(letterR)
contour(f)
contour(f, W=owin(c(1,5),c(-1,4)), eps=0.1)
```

`methods.kppm` *Methods for Cluster Point Process Models*

Description

These are methods for the class `"kppm"`.

Usage

```
## S3 method for class 'kppm'
coef(object, ...)
## S3 method for class 'kppm'
formula(x, ...)
## S3 method for class 'kppm'
print(x, ...)
## S3 method for class 'kppm'
terms(x, ...)
## S3 method for class 'kppm'
labels(object, ...)
```

Arguments

`x,object`	An object of class `"kppm"`, representing a fitted cluster point process model.
`...`	Arguments passed to other methods.

Details

These functions are methods for the generic commands `coef`, `formula`, `print`, `terms` and `labels` for the class `"kppm"`.

An object of class `"kppm"` represents a fitted cluster point process model. It is obtained from `kppm`.

The method `coef.kppm` returns the vector of *regression coefficients* of the fitted model. It does not return the clustering parameters.

Value

See the help files for the corresponding generic functions.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

See Also

`kppm`, `plot.kppm`, `predict.kppm`, `simulate.kppm`, `update.kppm`, `vcov.kppm`, `as.ppm.kppm`.

Examples

```
data(redwood)
fit <- kppm(redwood ~ x, "MatClust")
coef(fit)
formula(fit)
tf <- terms(fit)
labels(fit)
```

`methods.layered` *Methods for Layered Objects*

Description

Methods for geometrical transformations of layered objects (class `"layered"`).

Usage

```
  ## S3 method for class 'layered'
shift(X, vec=c(0,0), ...)

  ## S3 method for class 'layered'
rotate(X, ..., centre=NULL)

  ## S3 method for class 'layered'
affine(X, ...)
```

```
  ## S3 method for class 'layered'
reflect(X)

  ## S3 method for class 'layered'
flipxy(X)

  ## S3 method for class 'layered'
rescale(X, s, unitname)

  ## S3 method for class 'layered'
scalardilate(X, ...)
```

Arguments

X	Object of class "layered".
...	Arguments passed to the relevant methods when applying the operation to each layer of X.
s	Rescaling factor passed to the relevant method for rescale. May be missing.
vec	Shift vector (numeric vector of length 2).
centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).
unitname	Optional. New name for the unit of length. A value acceptable to the function unitname<-

Details

These are methods for the generic functions shift, rotate, reflect, affine, rescale, scalardilate and flipxy for the class of layered objects.

A layered object represents data that should be plotted in successive layers, for example, a background and a foreground. See layered.

Value

Another object of class "layered".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

layered

Examples

```
L <- layered(letterR, runifpoint(20, letterR))
plot(L)
plot(rotate(L, pi/4))
```

methods.linfun *Methods for Functions on Linear Network*

Description

Methods for the class "linfun" of functions on a linear network.

Usage

```
  ## S3 method for class 'linfun'
print(x, ...)

  ## S3 method for class 'linfun'
summary(object, ...)

  ## S3 method for class 'linfun'
plot(x, ..., L=NULL, main)

  ## S3 method for class 'linfun'
as.data.frame(x, ...)

  ## S3 method for class 'linfun'
as.owin(W, ...)

  ## S3 method for class 'linfun'
as.function(x, ...)
```

Arguments

x,object,W	A function on a linear network (object of class "linfun").
L	A linear network
...	Extra arguments passed to as.linim, plot.linim, plot.im or print.default, or arguments passed to x if it is a function.
main	Main title for plot.

Details

These are methods for the generic functions plot, print, summary as.data.frame and as.function, and for the **spatstat** generic function as.owin.

An object of class "linfun" represents a mathematical function that could be evaluated at any location on a linear network. It is essentially an R function with some extra attributes.

The method as.owin.linfun extracts the two-dimensional spatial window containing the linear network.

The method plot.linfun first converts the function to a pixel image using as.linim.linfun, then plots the image using plot.linim.

Note that a linfun function may have additional arguments, other than those which specify the location on the network (see linfun). These additional arguments may be passed to plot.linfun.

Value

For `print.linfun` and `summary.linfun` the result is `NULL`.

For `plot.linfun` the result is the same as for `plot.linim`.

For the conversion methods, the result is an object of the required type: `as.owin.linfun` returns an object of class `"owin"`, and so on.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

Examples

```
X <- runiflpp(3, simplenet)
f <- nnfun(X)
f
plot(f)
as.function(f)
as.owin(f)
head(as.data.frame(f))
```

---

methods.linim — *Methods for Images on a Linear Network*

Description

Methods for the class `"linim"` of functions on a linear network.

Usage

```
  ## S3 method for class 'linim'
print(x, ...)

  ## S3 method for class 'linim'
summary(object, ...)

  ## S3 method for class 'linim'
as.im(X, ...)

  ## S3 method for class 'linim'
as.data.frame(x, ...)

  ## S3 method for class 'linim'
shift(X, ...)

  ## S3 method for class 'linim'
scalardilate(X, f, ..., origin=NULL)

  ## S3 method for class 'linim'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)
```

Arguments

X,x,object	A pixel image on a linear network (object of class "linim").
...	Extra arguments passed to other methods.
f	Numeric. Scalar dilation factor.
mat	Numeric matrix representing the linear transformation.
vec	Numeric vector of length 2 specifying the shift vector.
origin	Character string determining a location that will be shifted to the origin. Options are "centroid", "midpoint" and "bottomleft". Partially matched.

Details

These are methods for the generic functions print, summary and as.data.frame, and the **spatstat** generic functions as.im, shift, scalardilate and affine.

An object of class "linfun" represents a pixel image defined on a linear network.

The method as.im.linim extracts the pixel values and returns a pixel image of class "im".

The method as.data.frame.linim returns a data frame giving spatial locations (in cartesian and network coordinates) and corresponding function values.

The methods shift.linim, scalardilate.linim and affine.linim apply geometric transformations to the pixels and the underlying linear network, without changing the pixel values.

Value

For print.linim the result is NULL.

The function summary.linim returns an object of class "summary.linim". In normal usage this summary is automatically printed by print.summary.linim.

For as.im.linim the result is an object of class "im".

For the geometric transformations shift.linim, scalardilate.linim and affine.linim, the result is another object of class "linim".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

Examples

```
M <- as.mask.psp(as.psp(simplenet))
Z <- as.im(function(x,y) {x-y}, W=M)
X <- linim(simplenet, Z)

X
shift(X, c(1,1))
scalardilate(X, 2)
head(as.data.frame(X))
```

methods.linnet *Methods for Linear Networks*

Description

These are methods for the class "linnet" of linear networks.

Usage

```
as.linnet(X, ...)

## S3 method for class 'linnet'
as.linnet(X, ..., sparse)

## S3 method for class 'linnet'
as.owin(W, ...)

## S3 method for class 'linnet'
as.psp(x, ..., fatal=TRUE)

## S3 method for class 'linnet'
nsegments(x)

## S3 method for class 'linnet'
nvertices(x, ...)

## S3 method for class 'linnet'
pixellate(x, ...)

## S3 method for class 'linnet'
print(x, ...)

## S3 method for class 'linnet'
summary(object, ...)

## S3 method for class 'linnet'
unitname(x)

## S3 replacement method for class 'linnet'
unitname(x) <- value

vertexdegree(x)

## S3 method for class 'linnet'
vertices(w)

## S3 method for class 'linnet'
volume(x)

## S3 method for class 'linnet'
Window(X, ...)
```

Arguments

x,X,object,w,W	An object of class "linnet" representing a linear network.
...	Arguments passed to other methods.
value	A valid name for the unit of length for x. See unitname.
fatal	Logical value indicating whether data in the wrong format should lead to an error (fatal=TRUE) or a warning (fatal=FALSE).
sparse	Logical value indicating whether to use a sparse matrix representation, as explained in linnet. Default is to keep the same representation as in X.

Details

The function as.linnet is generic. It converts data from some other format into an object of class "linnet". The method as.linnet.lpp extracts the linear network information from an lpp object.

The other functions are methods for the generic commands as.owin, as.psp, nsegments, nvertices, pixellate, print, summary, unitname, unitname<-, vertices, volume and Window for the class "linnet".

The methods as.owin.linnet and Window.linnet extract the window containing the linear network, and return it as an object of class "owin".

The method as.psp.linnet extracts the lines of the linear network as a line segment pattern (object of class "psp") while nsegments.linnet simply counts the number of line segments.

The method vertices.linnet extracts the vertices (nodes) of the linear network and nvertices.linnet simply counts the vertices. The function vertexdegree calculates the topological degree of each vertex (the number of lines emanating from that vertex) and returns these values as an integer vector.

The method pixellate.linnet applies as.psp.linnet to convert the network to a collection of line segments, then invokes pixellate.psp.

Value

For as.linnet the value is an object of class "linnet". For other functions, see the help file for the corresponding generic function.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

linnet.

Generic functions: as.owin, as.psp, nsegments, nvertices, pixellate, print, summary, unitname, unitname<-, vertices, volume and Window.

Special tools: thinNetwork, insertVertices, connected.linnet.

lixellate for dividing segments into shorter segments.

Examples

```
simplenet
summary(simplenet)
nsegments(simplenet)
nvertices(simplenet)
pixellate(simplenet)
```

```
volume(simplenet)
unitname(simplenet) <- c("cubit", "cubits")
Window(simplenet)
```

---

methods.lpp *Methods for Point Patterns on a Linear Network*

Description

These are methods specifically for the class "lpp" of point patterns on linear networks.

Usage

```
## S3 method for class 'lpp'
as.ppp(X, ..., fatal=TRUE)

## S3 method for class 'lpp'
as.psp(x, ..., fatal=TRUE)

## S3 replacement method for class 'lpp'
marks(x, ...) <- value

## S3 method for class 'lpp'
nsegments(x)

## S3 method for class 'lpp'
print(x, ...)

## S3 method for class 'summary.lpp'
print(x, ...)

## S3 method for class 'lpp'
summary(object, ...)

## S3 method for class 'lpp'
unitname(x)

## S3 replacement method for class 'lpp'
unitname(x) <- value

## S3 method for class 'lpp'
unmark(X)
```

Arguments

x,X,object	An object of class "lpp" representing a point pattern on a linear network.
...	Arguments passed to other methods.
value	Replacement value for the marks or unitname of x. See Details.
fatal	Logical value indicating whether data in the wrong format should lead to an error (fatal=TRUE) or a warning (fatal=FALSE).

Details

These are methods for the generic functions `as.ppp`, `as.psp`, `marks<-`, `nsegments`, `print`, `summary`, `unitname`, `unitname<-` and `unmark` for objects of the class "lpp".

For "marks<-.lpp" the replacement `value` should be either `NULL`, or a vector of length equal to the number of points in `x`, or a data frame with one row for each point in `x`.

For "unitname<-.lpp" the replacement `value` should be a valid name for the unit of length, as described in `unitname`.

Value

See the documentation on the corresponding generic function.

Other methods

An object of class "lpp" also inherits the class "ppx" for which many other methods are available. See `methods.ppx`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

`lpp`, `intensity.lpp`, `methods.ppx`

Examples

```
X <- runiflpp(10, simplenet)
X
as.ppp(X)
summary(X)
unitname(X) <- c("furlong", "furlongs")
```

`methods.lppm` *Methods for Fitted Point Process Models on a Linear Network*

Description

These are methods for the class "lppm" of fitted point process models on a linear network.

Usage

```
  ## S3 method for class 'lppm'
coef(object, ...)

  ## S3 method for class 'lppm'
emend(object, ...)

  ## S3 method for class 'lppm'
extractAIC(fit, ...)
```

```
## S3 method for class 'lppm'
formula(x, ...)

## S3 method for class 'lppm'
logLik(object, ...)

## S3 method for class 'lppm'
deviance(object, ...)

## S3 method for class 'lppm'
nobs(object, ...)

## S3 method for class 'lppm'
print(x, ...)

## S3 method for class 'lppm'
summary(object, ...)

## S3 method for class 'lppm'
terms(x, ...)

## S3 method for class 'lppm'
update(object, ...)

## S3 method for class 'lppm'
valid(object, ...)

## S3 method for class 'lppm'
vcov(object, ...)

## S3 method for class 'lppm'
as.linnet(X, ...)
```

Arguments

object,fit,x,X	An object of class "lppm" representing a fitted point process model on a linear network.
...	Arguments passed to other methods, usually the method for the class "ppm".

Details

These are methods for the generic commands coef, emend, extractAIC, formula, logLik, deviance, nobs, print, summary, terms, update, valid and vcov for the class "lppm".

Value

See the default methods.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

lppm, plot.lppm.

Examples

```
X <- runiflpp(15, simplenet)
fit <- lppm(X ~ x)
print(fit)
coef(fit)
formula(fit)
terms(fit)
logLik(fit)
deviance(fit)
nobs(fit)
extractAIC(fit)
update(fit, ~1)
valid(fit)
vcov(fit)
```

---

methods.objsurf   *Methods for Objective Function Surfaces*

Description

Methods for printing and plotting an objective function surface.

Usage

```
## S3 method for class 'objsurf'
print(x, ...)
## S3 method for class 'objsurf'
plot(x, ...)
## S3 method for class 'objsurf'
image(x, ...)
## S3 method for class 'objsurf'
contour(x, ...)
## S3 method for class 'objsurf'
persp(x, ...)
```

Arguments

x	Object of class "objsurf" representing an objective function surface.
...	Additional arguments passed to plot methods.

Details

These are methods for the generic functions print, plot, image, contour and persp for the class "objsurf".

Value

For `print.objsurf`, `plot.objsurf` and `image.objsurf` the value is `NULL`.

For `contour.objsurf` and `persp.objsurf` the value is described in the help for `contour.default` and `persp.default` respectively.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Ege Rubak <rubak@math.aau.dk>.

See Also

`objsurf`

Examples

```
fit <- kppm(redwood ~ 1, "Thomas")
os <- objsurf(fit)
 os
 plot(os)
 contour(os, add=TRUE)
 persp(os)
```

`methods.pp3` *Methods for three-dimensional point patterns*

Description

Methods for class "pp3".

Usage

```
  ## S3 method for class 'pp3'
print(x, ...)
  ## S3 method for class 'summary.pp3'
print(x, ...)
  ## S3 method for class 'pp3'
summary(object, ...)
  ## S3 method for class 'pp3'
unitname(x)
  ## S3 replacement method for class 'pp3'
unitname(x) <- value
```

Arguments

`x,object`	Object of class "pp3".
`...`	Ignored.
`value`	Name of the unit of length. See `unitname`.

Details

These are methods for the generic functions `print`, `summary`, `unitname` and `unitname<-` for the class `"pp3"` of three-dimensional point patterns.

The `print` and `summary` methods print a description of the point pattern.

The `unitname` method extracts the name of the unit of length in which the point coordinates are expressed. The `unitname<-` method assigns the name of the unit of length.

Value

For `print.pp3` the value is `NULL`. For `unitname.pp3` an object of class `"units"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`pp3`, `print`, `unitname` `unitname<-`

Examples

```
X <- pp3(runif(42),runif(42),runif(42), box3(c(0,1), unitname="mm"))
X
unitname(X)
unitname(X) <- c("foot", "feet")
summary(X)
```

`methods.ppx` *Methods for Multidimensional Space-Time Point Patterns*

Description

Methods for printing and plotting a general multidimensional space-time point pattern.

Usage

```
## S3 method for class 'ppx'
print(x, ...)
## S3 method for class 'ppx'
plot(x, ...)
## S3 method for class 'ppx'
unitname(x)
## S3 replacement method for class 'ppx'
unitname(x) <- value
```

Arguments

`x`	Multidimensional point pattern (object of class `"ppx"`).
`...`	Additional arguments passed to plot methods.
`value`	Name of the unit of length. See `unitname`.

Details

These are methods for the generic functions `print`, `plot`, `unitname` and `unitname<-` for the class `"ppx"` of multidimensional point patterns.

The `print` method prints a description of the point pattern and its spatial domain.

The `unitname` method extracts the name of the unit of length in which the point coordinates are expressed. The `unitname<-` method assigns the name of the unit of length.

Value

For `print.ppx` the value is `NULL`. For `unitname.ppx` an object of class `"units"`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`ppx`, `unitname`

`methods.rho2hat` *Methods for Intensity Functions of Two Spatial Covariates*

Description

These are methods for the class `"rho2hat"`.

Usage

```
## S3 method for class 'rho2hat'
plot(x, ..., do.points=FALSE)

## S3 method for class 'rho2hat'
print(x, ...)

## S3 method for class 'rho2hat'
predict(object, ..., relative=FALSE)
```

Arguments

`x,object`	An object of class `"rho2hat"`.
`...`	Arguments passed to other methods.
`do.points`	Logical value indicating whether to plot the observed values of the covariates at the data points.
`relative`	Logical value indicating whether to compute the estimated point process intensity (`relative=FALSE`) or the relative risk (`relative=TRUE`) in the case of a relative risk estimate.

Details

These functions are methods for the generic commands print, predict and plot for the class "rho2hat".

An object of class "rho2hat" is an estimate of the intensity of a point process, as a function of two given spatial covariates. See rho2hat.

The method plot.rho2hat displays the estimated function ρ using plot.fv, and optionally adds a rug plot of the observed values of the covariate. In this plot the two axes represent possible values of the two covariates.

The method predict.rho2hat computes a pixel image of the intensity $\rho(Z_1(u), Z_2(u))$ at each spatial location u, where $Z_1(u)$ and $Z_2(u)$ are the two spatial covariates.

Value

For predict.rho2hat the value is a pixel image (object of class "im"). For other functions, the value is NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

rho2hat

Examples

```
r2 <- with(bei.extra, rho2hat(bei, elev, grad))
r2
plot(r2)
plot(predict(r2))
```

---

methods.rhohat *Methods for Intensity Functions of Spatial Covariate*

Description

These are methods for the class "rhohat".

Usage

```
## S3 method for class 'rhohat'
print(x, ...)

## S3 method for class 'rhohat'
plot(x, ..., do.rug=TRUE)

## S3 method for class 'rhohat'
predict(object, ..., relative=FALSE,
             what=c("rho", "lo", "hi", "se"))

## S3 method for class 'rhohat'
simulate(object, nsim=1, ..., drop=TRUE)
```

Arguments

x,object	An object of class "rhohat" representing a smoothed estimate of the intensity function of a point process.
...	Arguments passed to other methods.
do.rug	Logical value indicating whether to plot the observed values of the covariate as a rug plot along the horizontal axis.
relative	Logical value indicating whether to compute the estimated point process intensity (relative=FALSE) or the relative risk (relative=TRUE) in the case of a relative risk estimate.
nsim	Number of simulations to be generated.
drop	Logical value indicating what to do when nsim=1. If drop=TRUE (the default), a point pattern is returned. If drop=FALSE, a list of length 1 containing a point pattern is returned.
what	Optional character string (partially matched) specifying which value should be calculated: either the function estimate (what="rho", the default), the lower or upper end of the confidence interval (what="lo" or what="hi") or the standard error (what="se").

Details

These functions are methods for the generic commands print, plot, predict and simulate for the class "rhohat".

An object of class "rhohat" is an estimate of the intensity of a point process, as a function of a given spatial covariate. See rhohat.

The method plot.rhohat displays the estimated function ρ using plot.fv, and optionally adds a rug plot of the observed values of the covariate.

The method predict.rhohat computes a pixel image of the intensity $\rho(Z(u))$ at each spatial location u, where Z is the spatial covariate.

The method simulate.rhohat invokes predict.rhohat to determine the predicted intensity, and then simulates a Poisson point process with this intensity.

Value

For predict.rhohat the value is a pixel image (object of class "im" or "linim"). For simulate.rhohat the value is a point pattern (object of class "ppp" or "lpp"). For other functions, the value is NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

rhohat

Examples

```
X <-  rpoispp(function(x,y){exp(3+3*x)})
rho <- rhohat(X, function(x,y){x})
rho
plot(rho)
```

```
Y <- predict(rho)
plot(Y)
plot(simulate(rho), add=TRUE)
#
fit <- ppm(X, ~x)
rho <- rhohat(fit, "y")
opa <- par(mfrow=c(1,2))
plot(predict(rho))
plot(predict(rho, relative=TRUE))
par(opa)
plot(predict(rho, what="se"))
```

`methods.slrm` *Methods for Spatial Logistic Regression Models*

Description

These are methods for the class "slrm".

Usage

```
## S3 method for class 'slrm'
formula(x, ...)
## S3 method for class 'slrm'
print(x, ...)
## S3 method for class 'slrm'
terms(x, ...)
## S3 method for class 'slrm'
labels(object, ...)
## S3 method for class 'slrm'
update(object, ..., evaluate = TRUE, env = parent.frame())
```

Arguments

x,object	An object of class "slrm", representing a fitted spatial logistic regression model.
...	Arguments passed to other methods.
evaluate	Logical value. If TRUE, evaluate the updated call to slrm, so that the model is refitted; if FALSE, simply return the updated call.
env	Optional environment in which the model should be updated.

Details

These functions are methods for the generic commands formula, update, print, terms and labels for the class "slrm".

An object of class "slrm" represents a fitted spatial logistic regression model. It is obtained from slrm.

Value

See the help files for the corresponding generic functions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

slrm, plot.slrm, predict.slrm, simulate.slrm, vcov.slrm, coef.slrm.

Examples

```
data(redwood)
fit <- slrm(redwood ~ x)
coef(fit)
formula(fit)
tf <- terms(fit)
labels(fit)
```

---

methods.ssf — *Methods for Spatially Sampled Functions*

Description

Methods for various generic commands, for the class "ssf" of spatially sampled functions.

Usage

```
  ## S3 method for class 'ssf'
marks(x, ...)

  ## S3 replacement method for class 'ssf'
marks(x, ...) <- value

  ## S3 method for class 'ssf'
unmark(X)

  ## S3 method for class 'ssf'
as.im(X, ...)

  ## S3 method for class 'ssf'
as.function(x, ...)

  ## S3 method for class 'ssf'
as.ppp(X, ...)

  ## S3 method for class 'ssf'
print(x, ..., brief=FALSE)

  ## S3 method for class 'ssf'
range(x, ...)

  ## S3 method for class 'ssf'
min(x, ...)
```

```
  ## S3 method for class 'ssf'
max(x, ...)

  ## S3 method for class 'ssf'
integral(f, domain=NULL, ..., weights=attr(f, "weights"))
```

Arguments

`x,X,f`	A spatially sampled function (object of class "ssf").
`...`	Arguments passed to the default method.
`brief`	Logical value controlling the amount of detail printed.
`value`	Matrix of replacement values for the function.
`domain`	Optional. Domain of integration. An object of class"owin".
`weights`	Optional. Numeric vector of weights associated with the sample points.

Details

An object of class "ssf" represents a function (real- or vector-valued) that has been sampled at a finite set of points.

The commands documented here are methods for this class, for the generic commands `marks`, `marks<-`, `unmark`, `as.im`, `as.function`, `as.ppp`, `print`, `range`, `min`, `max` and `integral`.

Value

`marks` returns a matrix.

`marks(x) <- value` returns an object of class "ssf".

`as.owin` returns a window (object of class "owin").

`as.ppp` and `unmark` return a point pattern (object of class "ppp").

`as.function` returns a `function(x,y)` of class "funxy".

`print` returns `NULL`.

`range` returns a numeric vector of length 2. `min` and `max` return a single numeric value.

`integral` returns a numeric value (if `x` had numeric values) or a numeric vector (if `x` had vector values).

Author(s)

Adrian Baddeley

See Also

`ssf`

Examples

```
X <- cells[1:4]
f <- ssf(X, nndist(X, k=1:3))
f
marks(f)
as.ppp(f)
as.im(f)
```

methods.unitname *Methods for Units*

Description

Methods for class "unitname".

Usage

```
  ## S3 method for class 'unitname'
print(x, ...)
  ## S3 method for class 'unitname'
summary(object, ...)
  ## S3 method for class 'unitname'
rescale(X, s, unitname)
  ## S3 method for class 'unitname'
compatible(A,B, ..., coerce=TRUE)
  ## S3 method for class 'unitname'
harmonise(..., coerce=TRUE, single=FALSE)
  ## S3 method for class 'unitname'
harmonize(..., coerce=TRUE, single=FALSE)
```

Arguments

x,X,A,B,object	Objects of class "unitname" representing units of length.
...	Other arguments. For print.unitname these arguments are passed to print.default. For summary.unitname they are ignored. For compatible.unitname and harmonise.unitname these arguments are other objects of class "unitname".
s	Conversion factor: the new units are s times the old units.
unitname	Optional new name for the unit. If present, this overrides the rescaling operation and simply substitutes the new name for the old one.
coerce	Logical. If TRUE, a null unit of length is compatible with any non-null unit.
single	Logical value indicating whether to return a single unitname, or a list of unitnames.

Details

These are methods for the generic functions print, summary, rescale and compatible for the class "unitname".

An object of class "unitname" represents a unit of length.

The print method prints a description of the unit of length, and the summary method gives a more detailed description.

The rescale method changes the unit of length by rescaling it.

The compatible method tests whether two or more units of length are compatible.

The harmonise method returns the common unit of length if there is one. For consistency with other methods for harmonise, the result is a list of unitname objects, with one entry for each argument in .... All of these entries are identical. This can be overridden by setting single=TRUE when the result will be a single unitname object.

Value

For `print.unitname` the value is `NULL`. For `summary.unitname` the value is an object of class `summary.unitname` (with its own print method). For `rescale.unitname` the value is another object of class `"unitname"`. For `compatible.unitname` the result is logical. For `harmonise.unitname` the result is a list of identical unitnames if `single=FALSE` (the default), or a single unitname if `single=TRUE`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`box3`, `print`, `unitname`

`methods.zclustermodel` *Methods for Cluster Models*

Description

Methods for the experimental class of cluster models.

Usage

```
 ## S3 method for class 'zclustermodel'
pcfmodel(model, ...)

 ## S3 method for class 'zclustermodel'
predict(object, ...,
                  locations, type = "intensity", ngrid = NULL)

 ## S3 method for class 'zclustermodel'
print(x, ...)
```

Arguments

`model,object,x`	Object of class `"zclustermodel"`.
`...`	Arguments passed to other methods.
`locations`	Locations where prediction should be performed. A window or a point pattern.
`type`	Currently must equal `"intensity"`.
`ngrid`	Pixel grid dimensions for prediction, if `locations` is a rectangle or polygon.

Details

Experimental.

Value

Same as for other methods.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

zclustermodel

Examples

```
m <- zclustermodel("Thomas", kappa=10, mu=5, scale=0.1)
m2 <- zclustermodel("VarGamma", kappa=10, mu=10, scale=0.1, nu=0.7)
m
m2
g <- pcfmodel(m)
g(0.2)
g2 <- pcfmodel(m2)
g2(1)
Z <- predict(m, locations=square(2))
Z2 <- predict(m2, locations=square(1))
varcount(m, square(1))
varcount(m2, square(1))
```

---

midpoints.psp *Midpoints of Line Segment Pattern*

Description

Computes the midpoints of each line segment in a line segment pattern.

Usage

```
midpoints.psp(x)
```

Arguments

x	A line segment pattern (object of class "psp").

Details

The midpoint of each line segment is computed.

Value

Point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

summary.psp, lengths.psp, angles.psp

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
b <- midpoints.psp(a)
```

---

mincontrast *Method of Minimum Contrast*

Description

A general low-level algorithm for fitting theoretical point process models to point pattern data by the Method of Minimum Contrast.

Usage

```
mincontrast(observed, theoretical, startpar, ...,
          ctrl=list(q = 1/4, p = 2, rmin=NULL, rmax=NULL),
          fvlab=list(label=NULL, desc="minimum contrast fit"),
          explain=list(dataname=NULL, modelname=NULL, fname=NULL),
  adjustment=NULL)
```

Arguments

`observed`	Summary statistic, computed for the data. An object of class `"fv"`.
`theoretical`	An R language function that calculates the theoretical expected value of the summary statistic, given the model parameters. See Details.
`startpar`	Vector of initial values of the parameters of the point process model (passed to `theoretical`).
`...`	Additional arguments passed to the function `theoretical` and to the optimisation algorithm `optim`.
`ctrl`	Optional. List of arguments controlling the optimisation. See Details.
`fvlab`	Optional. List containing some labels for the return value. See Details.
`explain`	Optional. List containing strings that give a human-readable description of the model, the data and the summary statistic.
`adjustment`	Internal use only.

Details

This function is a general algorithm for fitting point process models by the Method of Minimum Contrast. If you want to fit the Thomas process, see `thomas.estK`. If you want to fit a log-Gaussian Cox process, see `lgcp.estK`. If you want to fit the Matern cluster process, see `matclust.estK`.

The Method of Minimum Contrast (Diggle and Gratton, 1984) is a general technique for fitting a point process model to point pattern data. First a summary function (typically the K function) is computed from the data point pattern. Second, the theoretical expected value of this summary statistic under the point process model is derived (if possible, as an algebraic expression involving the parameters of the model) or estimated from simulations of the model. Then the model is fitted by finding the optimal parameter values for the model to give the closest match between the theoretical and empirical curves.

The argument observed should be an object of class "fv" (see fv.object) containing the values of a summary statistic computed from the data point pattern. Usually this is the function $K(r)$ computed by Kest or one of its relatives.

The argument theoretical should be a user-supplied function that computes the theoretical expected value of the summary statistic. It must have an argument named par that will be the vector of parameter values for the model (the length and format of this vector are determined by the starting values in startpar). The function theoretical should also expect a second argument (the first argument other than par) containing values of the distance r for which the theoretical value of the summary statistic $K(r)$ should be computed. The value returned by theoretical should be a vector of the same length as the given vector of r values.

The argument ctrl determines the contrast criterion (the objective function that will be minimised). The algorithm minimises the criterion

$$D(\theta) = \int_{r_{\mbox{min}}}^{r_{\mbox{max}}} |\hat{F}(r)^q - F_\theta(r)^q|^p \, \mathrm{d}r$$

where θ is the vector of parameters of the model, $\hat{F}(r)$ is the observed value of the summary statistic computed from the data, $F_\theta(r)$ is the theoretical expected value of the summary statistic, and p, q are two exponents. The default is q = 1/4, p=2 so that the contrast criterion is the integrated squared difference between the fourth roots of the two functions (Waagepetersen, 2006).

The other arguments just make things print nicely. The argument fvlab contains labels for the component fit of the return value. The argument explain contains human-readable strings describing the data, the model and the summary statistic.

The "..." argument of mincontrast can be used to pass extra arguments to the function theoretical and/or to the optimisation function optim. In this case, the function theoretical should also have a "..." argument and should ignore it (so that it ignores arguments intended for optim).

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following components:

par	Vector of fitted parameter values.
fit	Function value table (object of class "fv") containing the observed values of the summary statistic (observed) and the theoretical values of the summary statistic computed from the fitted model parameters.
opt	The return value from the optimizer optim.
crtl	The control parameters of the algorithm.
info	List of explanatory strings.

Author(s)

Rasmus Waagepetersen <rw@math.auc.dk>, adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu

References

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Waagepetersen, R. (2006). An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63** (2007) 252–258.

See Also

`kppm`, `lgcp.estK`, `matclust.estK`, `thomas.estK`,

`MinkowskiSum` *Minkowski Sum of Windows*

Description

Compute the Minkowski sum of two spatial windows.

Usage

```
MinkowskiSum(A, B)

A %(+)% B

dilationAny(A, B)
```

Arguments

`A,B`	Windows (objects of class "owin"), point patterns (objects of class "ppp") or line segment patterns (objects of class "psp") in any combination.

Details

The operator `A %(+)% B` and function `MinkowskiSum(A,B)` are synonymous: they both compute the Minkowski sum of the windows `A` and `B`. The function `dilationAny` computes the Minkowski dilation `A %(+)% reflect(B)`.

The Minkowski sum of two spatial regions A and B is another region, formed by taking all possible pairs of points, one in A and one in B, and adding them as vectors. The Minkowski Sum $A \oplus B$ is the set of all points $a + b$ where a is in A and b is in B. A few common facts about the Minkowski sum are:

- The sum is symmetric: $A \oplus B = B \oplus A$.
- If B is a single point, then $A \oplus B$ is a shifted copy of A.
- If A is a square of side length a, and B is a square of side length b, with sides that are parallel to the coordinate axes, then $A \oplus B$ is a square of side length $a + b$.
- If A and B are discs of radius r and s respectively, then $A \oplus B$ is a disc of redius $r + s$.
- If B is a disc of radius r centred at the origin, then $A \oplus B$ is equivalent to the *morphological dilation* of A by distance r. See `dilation`.

The Minkowski dilation is the closely-related region $A \oplus (-B)$ where $(-B)$ is the reflection of B through the origin. The Minkowski dilation is the set of all vectors z such that, if B is shifted by z, the resulting set $B + z$ has nonempty intersection with A.

The algorithm currently computes the result as a polygonal window using the **polyclip** library. It will be quite slow if applied to binary mask windows.

The arguments `A` and `B` can also be point patterns or line segment patterns. These are interpreted as spatial regions, the Minkowski sum is computed, and the result is returned as an object of the most appropriate type. The Minkowski sum of two point patterns is another point pattern. The Minkowski sum of a point pattern and a line segment pattern is another line segment pattern.

Value

A window (object of class "owin") except that if A is a point pattern, then the result is an object of the same type as B (and vice versa).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

dilation, erosionAny

Examples

```
B <- square(0.2)
RplusB <- letterR %(+)% B

opa <- par(mfrow=c(1,2))
FR <- grow.rectangle(Frame(letterR), 0.3)
plot(FR, main="")
plot(letterR, add=TRUE, lwd=2, hatch=TRUE, hatchargs=list(texture=5))
plot(shift(B, vec=c(3.675, 3)),
     add=TRUE, border="red", lwd=2)
plot(FR, main="")
plot(letterR, add=TRUE, lwd=2, hatch=TRUE, hatchargs=list(texture=5))
plot(RplusB, add=TRUE, border="blue", lwd=2,
       hatch=TRUE, hatchargs=list(col="blue"))
par(opa)

plot(cells %(+)% square(0.1))
```

---

miplot *Morisita Index Plot*

Description

Displays the Morisita Index Plot of a spatial point pattern.

Usage

```
miplot(X, ...)
```

Arguments

X	A point pattern (object of class "ppp") or something acceptable to as.ppp.
...	Optional arguments to control the appearance of the plot.

Details

Morisita (1959) defined an index of spatial aggregation for a spatial point pattern based on quadrat counts. The spatial domain of the point pattern is first divided into Q subsets (quadrats) of equal size and shape. The numbers of points falling in each quadrat are counted. Then the Morisita Index is computed as

$$\mathrm{MI} = Q \frac{\sum_{i=1}^{Q} n_i(n_i - 1)}{N(N-1)}$$

where n_i is the number of points falling in the i-th quadrat, and N is the total number of points. If the pattern is completely random, `MI` should be approximately equal to 1. Values of `MI` greater than 1 suggest clustering.

The *Morisita Index plot* is a plot of the Morisita Index `MI` against the linear dimension of the quadrats. The point pattern dataset is divided into 2×2 quadrats, then 3×3 quadrats, etc, and the Morisita Index is computed each time. This plot is an attempt to discern different scales of dependence in the point pattern data.

Value

None.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

M. Morisita (1959) Measuring of the dispersion of individuals and analysis of the distributional patterns. Memoir of the Faculty of Science, Kyushu University, Series E: Biology. **2**: 215–235.

See Also

quadratcount

Examples

```
data(longleaf)
miplot(longleaf)
opa <- par(mfrow=c(2,3))
data(cells)
data(japanesepines)
data(redwood)
plot(cells)
plot(japanesepines)
plot(redwood)
miplot(cells)
miplot(japanesepines)
miplot(redwood)
par(opa)
```

`model.depends` *Identify Covariates Involved in each Model Term*

Description

Given a fitted model (of any kind), identify which of the covariates is involved in each term of the model.

Usage

```
model.depends(object)
model.is.additive(object)
model.covariates(object, fitted=TRUE, offset=TRUE)
has.offset.term(object)
has.offset(object)
```

Arguments

`object`	A fitted model of any kind.
`fitted,offset`	Logical values determining which type of covariates to include.

Details

The `object` can be a fitted model of any kind, including models of the classes `lm`, `glm` and `ppm`.

To be precise, `object` must belong to a class for which there are methods for `formula`, `terms` and `model.matrix`.

The command `model.depends` determines the relationship between the original covariates (the data supplied when `object` was fitted) and the canonical covariates (the columns of the design matrix). It returns a logical matrix, with one row for each canonical covariate, and one column for each of the original covariates, with the `i,j` entry equal to `TRUE` if the `i`th canonical covariate depends on the `j`th original covariate.

If the model formula of `object` includes offset terms (see `offset`), then the return value of `model.depends` also has an attribute `"offset"`. This is a logical value or matrix with one row for each offset term and one column for each of the original covariates, with the `i,j` entry equal to `TRUE` if the `i`th offset term depends on the `j`th original covariate.

The command `model.covariates` returns a character vector containing the names of all (original) covariates that were actually used to fit the model. By default, this includes all covariates that appear in the model formula, including offset terms as well as canonical covariate terms. To omit the offset terms, set `offset=FALSE`. To omit the canonical covariate terms, set `fitted=FALSE`.

The command `model.is.additive` determines whether the model is additive, in the sense that there is no canonical covariate that depends on two or more original covariates. It returns a logical value.

The command `has.offset.term` is a faster way to determine whether the model *formula* includes an `offset` term.

The functions `model.depends` and `has.offset.term` only detect `offset` terms which are present in the model formula. They do not detect numerical offsets in the model object, that were inserted using the `offset` argument in `lm`, `glm` etc. To detect the presence of offsets of both kinds, use `has.offset`.

Value

A logical value or matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, model.matrix

Examples

```
x <- 1:10
y <- 3*x + 2
z <- rep(c(-1,1), 5)
fit <- lm(y ~ poly(x,2) + sin(z))
model.depends(fit)
model.covariates(fit)
model.is.additive(fit)

fitoff1 <- lm(y ~ x + offset(z))
fitoff2 <- lm(y ~ x, offset=z)
has.offset.term(fitoff1)
has.offset(fitoff1)
has.offset.term(fitoff2)
has.offset(fitoff2)
```

---

model.frame.ppm *Extract the Variables in a Point Process Model*

Description

Given a fitted point process model, this function returns a data frame containing all the variables needed to fit the model using the Berman-Turner device.

Usage

```
## S3 method for class 'ppm'
model.frame(formula, ...)

## S3 method for class 'kppm'
model.frame(formula, ...)

## S3 method for class 'dppm'
model.frame(formula, ...)

## S3 method for class 'lppm'
model.frame(formula, ...)
```

Arguments

formula A fitted point process model. An object of class "ppm" or "kppm" or "dppm" or "lppm".

... Additional arguments passed to model.frame.glm.

Details

The function model.frame is generic. These functions are method for model.frame for fitted point process models (objects of class "ppm" or "kppm" or "dppm" or "lppm").

The first argument should be a fitted point process model; it has to be named formula for consistency with the generic function.

The result is a data frame containing all the variables used in fitting the model. The data frame has one row for each quadrature point used in fitting the model. The quadrature scheme can be extracted using quad.ppm.

Value

A data.frame containing all the variables used in the fitted model, plus additional variables specified in .... It has an additional attribute "terms" containing information about the model formula. For details see model.frame.glm.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

See Also

ppm, kppm, dppm, lppm, model.frame, model.matrix.ppm

Examples

```
fit <- ppm(cells ~ x)
mf <- model.frame(fit)
kfit <- kppm(redwood ~ x, "Thomas")
kmf <- model.frame(kfit)
```

model.images *Compute Images of Constructed Covariates*

Description

For a point process model fitted to spatial point pattern data, this function computes pixel images of the covariates in the design matrix.

Usage

```
model.images(object, ...)

## S3 method for class 'ppm'
model.images(object, W = as.owin(object), ...)

## S3 method for class 'kppm'
model.images(object, W = as.owin(object), ...)

## S3 method for class 'dppm'
model.images(object, W = as.owin(object), ...)

## S3 method for class 'lppm'
model.images(object, L = as.linnet(object), ...)

## S3 method for class 'slrm'
model.images(object, ...)
```

Arguments

`object`	The fitted point process model. An object of class "ppm" or "kppm" or "lppm" or "slrm" or "dppm".
`W`	A window (object of class "owin") in which the images should be computed. Defaults to the window in which the model was fitted.
`L`	A linear network (object of class "linnet") in which the images should be computed. Defaults to the network in which the model was fitted.
`...`	Other arguments (such as `na.action`) passed to `model.matrix.lm`.

Details

This command is similar to `model.matrix.ppm` except that it computes pixel images of the covariates, instead of computing the covariate values at certain points only.

The `object` must be a fitted spatial point process model object of class "ppm" (produced by the model-fitting function `ppm`) or class "kppm" (produced by the fitting function `kppm`) or class "dppm" (produced by the fitting function `dppm`) or class "lppm" (produced by `lppm`) or class "slrm" (produced by `slrm`).

The spatial covariates required by the model-fitting procedure are computed at every pixel location in the window `W`. For `lppm` objects, the covariates are computed at every location on the network `L`. For `slrm` objects, the covariates are computed on the pixels that were used to fit the model.

Note that the spatial covariates computed here are not the original covariates that were supplied when fitting the model. Rather, they are the covariates that actually appear in the loglinear representation of the (conditional) intensity and in the columns of the design matrix. For example, they might include dummy or indicator variables for different levels of a factor, depending on the contrasts that are in force.

The pixel resolution is determined by `W` if `W` is a mask (that is `W$type = "mask"`). Otherwise, the pixel resolution is determined by `spatstat.options`.

The format of the result depends on whether the original point pattern data were marked or unmarked.

- If the original dataset was unmarked, the result is a named list of pixel images (objects of class `"im"`) containing the values of the spatial covariates. The names of the list elements are the names of the covariates determined by `model.matrix.lm`. The result is also of class `"solist"` so that it can be plotted immediately.
- If the original dataset was a multitype point pattern, the result is a `hyperframe` with one column for each possible type of points. Each column is a named list of pixel images (objects of class `"im"`) containing the values of the spatial covariates. The row names of the hyperframe are the names of the covariates determined by `model.matrix.lm`.

Value

A list (of class `"solist"`) or array (of class `"hyperframe"`) containing pixel images (objects of class `"im"`). For `model.images.lppm`, the images are also of class `"linim"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`model.matrix.ppm`, `model.matrix`, `ppm`, `ppm.object`, `lppm`, `dppm`, `kppm`, `slrm`, `im`, `im.object`, `plot.solist`, `spatstat.options`

Examples

```
fit <- ppm(cells ~ x)
model.images(fit)
B <- owin(c(0.2, 0.4), c(0.3, 0.8))
model.images(fit, B)
fit2 <- ppm(cells ~ cut(x,3))
model.images(fit2)
fit3 <- slrm(japanesepines ~ x)
model.images(fit3)
fit4 <- ppm(amacrine ~ marks + x)
model.images(fit4)
```

model.matrix.mppm — *Extract Design Matrix of Point Process Model for Several Point Patterns*

Description

Given a point process model fitted to a list of point patterns, this function extracts the design matrix.

Usage

```
## S3 method for class 'mppm'
model.matrix(object, ..., keepNA=TRUE, separate=FALSE)
```

Arguments

object	A point process model fitted to several point patterns. An object of class "mppm".
...	Other arguments (such as na.action) passed to model.matrix.lm.
keepNA	Logical. Determines whether rows containing NA values will be deleted or retained.
separate	Logical value indicating whether to split the model matrix into sub-matrices corresponding to each of the original point patterns.

Details

This command is a method for the generic function model.matrix. It extracts the design matrix of a point process model fitted to several point patterns.

The argument object must be a fitted point process model (object of class "mppm") produced by the fitting algorithm mppm). This represents a point process model that has been fitted to a list of several point pattern datasets. See mppm for information.

The result is a matrix with one column for every constructed covariate in the model, and one row for every quadrature point.

If separate=TRUE this matrix will be split into sub-matrices corresponding to the original point patterns, and the result will be a list containing these matrices.

Value

A matrix (or list of matrices). Columns of the matrix are canonical covariates in the model.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

model.matrix, mppm.

Examples

```
fit <- mppm(Points ~ Image + x, demohyper)
head(model.matrix(fit))
# matrix with three columns: '(Intercept)', 'x' and 'Image'
```

model.matrix.ppm *Extract Design Matrix from Point Process Model*

Description

Given a point process model that has been fitted to spatial point pattern data, this function extracts the design matrix of the model.

Usage

```
   ## S3 method for class 'ppm'
model.matrix(object,
                              data=model.frame(object, na.action=NULL),
                              ...,
                              Q=NULL, keepNA=TRUE)

   ## S3 method for class 'kppm'
model.matrix(object,
                              data=model.frame(object, na.action=NULL),
                              ...,
                              Q=NULL, keepNA=TRUE)

   ## S3 method for class 'dppm'
model.matrix(object,
                              data=model.frame(object, na.action=NULL),
                              ...,
                              Q=NULL, keepNA=TRUE)

   ## S3 method for class 'lppm'
model.matrix(object,
                              data=model.frame(object, na.action=NULL),
                              ...,
                              keepNA=TRUE)

   ## S3 method for class 'ippm'
model.matrix(object,
                              data=model.frame(object, na.action=NULL),
                              ...,
                              Q=NULL, keepNA=TRUE,
      irregular=FALSE)
```

Arguments

object	The fitted point process model. An object of class "ppm" or "kppm" or "dppm" or "ippm" or "lppm".
data	A model frame, containing the data required for the Berman-Turner device.
Q	A point pattern (class "ppp") or quadrature scheme (class "quad") specifying new locations where the covariates should be computed.
keepNA	Logical. Determines whether rows containing NA values will be deleted or retained.

...	Other arguments (such as na.action) passed to model.matrix.lm.
irregular	Logical value indicating whether to include the irregular score components.

Details

These commands are methods for the generic function model.matrix. They extract the design matrix of a spatial point process model (class "ppm" or "kppm" or "dppm" or "lppm").

More precisely, this command extracts the design matrix of the generalised linear model associated with a spatial point process model.

The object must be a fitted point process model (object of class "ppm" or "kppm" or "dppm" or "lppm") fitted to spatial point pattern data. Such objects are produced by the model-fitting functions ppm, kppm, dppm and lppm.

The methods model.matrix.ppm, model.matrix.kppm, model.matrix.dppm and model.matrix.lppm extract the model matrix for the GLM.

The result is a matrix, with one row for every quadrature point in the fitting procedure, and one column for every constructed covariate in the design matrix.

If there are NA values in the covariates, the argument keepNA determines whether to retain or delete the corresponding rows of the model matrix. The default keepNA=TRUE is to retain them. Note that this differs from the default behaviour of many other methods for model.matrix, which typically delete rows containing NA.

The quadrature points themselves can be extracted using quad.ppm.

Value

A matrix. Columns of the matrix are canonical covariates in the model. Rows of the matrix correspond to quadrature points in the fitting procedure (provided keepNA=TRUE).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

model.matrix, model.images, ppm, kppm, dppm, lppm, ippm, ppm.object, quad.ppm, residuals.ppm

Examples

```
fit <- ppm(cells ~ x)
head(model.matrix(fit))
model.matrix(fit, Q=runifpoint(5))
kfit <- kppm(redwood ~ x, "Thomas")
m <- model.matrix(kfit)
```

model.matrix.slrm *Extract Design Matrix from Spatial Logistic Regression Model*

Description

This function extracts the design matrix of a spatial logistic regression model.

Usage

```
## S3 method for class 'slrm'
model.matrix(object, ..., keepNA=TRUE)
```

Arguments

object	A fitted spatial logistic regression model. An object of class "slrm".
...	Other arguments (such as na.action) passed to model.matrix.lm.
keepNA	Logical. Determines whether rows containing NA values will be deleted or retained.

Details

This command is a method for the generic function model.matrix. It extracts the design matrix of a spatial logistic regression.

The object must be a fitted spatial logistic regression (object of class "slrm"). Such objects are produced by the model-fitting function slrm.

Usually the result is a matrix with one column for every constructed covariate in the model, and one row for every pixel in the grid used to fit the model.

If object was fitted using split pixels (by calling slrm using the argument splitby) then the matrix has one row for every pixel or half-pixel.

Value

A matrix. Columns of the matrix are canonical covariates in the model.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

model.matrix, model.images, slrm.

Examples

```
fit <- slrm(japanesepines ~x)
head(model.matrix(fit))
# matrix with two columns: '(Intercept)' and 'x'
```

mppm *Fit Point Process Model to Several Point Patterns*

Description

Fits a Gibbs point process model to several point patterns simultaneously.

Usage

```
mppm(formula, data, interaction=Poisson(), ...,
     iformula=NULL,
     random=NULL,
     use.gam = FALSE,
     reltol.pql=1e-3,
     gcontrol=list())
```

Arguments

`formula`	A formula describing the systematic part of the model. Variables in the formula are names of columns in `data`.
`data`	A hyperframe (object of class `"hyperframe"`, see `hyperframe`) containing the point pattern responses and the explanatory variables.
`interaction`	Interpoint interaction(s) appearing in the model. Either an object of class `"interact"` describing the point process interaction structure, or a hyperframe (with the same number of rows as `data`) whose entries are objects of class `"interact"`.
`...`	Arguments passed to `ppm` controlling the fitting procedure.
`iformula`	Optional. A formula (with no left hand side) describing the interaction to be applied to each case. Each variable name in the formula should either be the name of a column in the hyperframe `interaction`, or the name of a column in the hyperframe `data` that is a vector or factor.
`random`	Optional. A formula (with no left hand side) describing a random effect. Variable names in the formula may be any of the column names of `data` and `interaction`. The formula must be recognisable to `lme`.
`use.gam`	Logical flag indicating whether to fit the model using `gam` or `glm`.
`reltol.pql`	Relative tolerance for successive steps in the penalised quasi-likelihood algorithm, used when the model includes random effects. The algorithm terminates when the root mean square of the relative change in coefficients is less than `reltol.pql`.
`gcontrol`	List of arguments to control the fitting algorithm. Arguments are passed to `glm.control` or `gam.control` or `lmeControl` depending on the kind of model being fitted. If the model has random effects, the arguments are passed to `lmeControl`. Otherwise, if `use.gam=TRUE` the arguments are passed to `gam.control`, and if `use.gam=FALSE` (the default) they are passed to `glm.control`.

Details

This function fits a common point process model to a dataset containing several different point patterns.

It extends the capabilities of the function `ppm` to deal with data such as

- replicated observations of spatial point patterns
- two groups of spatial point patterns
- a designed experiment in which the response from each unit is a point pattern.

The syntax of this function is similar to that of standard R model-fitting functions like `lm` and `glm`. The first argument `formula` is an R formula describing the systematic part of the model. The second argument `data` contains the responses and the explanatory variables. Other arguments determine the stochastic structure of the model.

Schematically, the data are regarded as the results of a designed experiment involving n experimental units. Each unit has a 'response', and optionally some 'explanatory variables' (covariates) describing the experimental conditions for that unit. In this context, *the response from each unit is a point pattern*. The value of a particular covariate for each unit can be either a single value (numerical, logical or factor), or a spatial covariate. A 'spatial' covariate is a quantity that depends on spatial location, for example, the soil acidity or altitude at each location. For the purposes of `mppm`, a spatial covariate must be stored as a pixel image (object of class `"im"`) which gives the values of the covariate at a fine grid of locations.

The argument `data` is a hyperframe (a generalisation of a data frame, see `hyperframe`). This is like a data frame except that the entries can be objects of any class. The hyperframe has one row for each experimental unit, and one column for each variable (response or explanatory variable).

The `formula` should be an R formula. The left hand side of `formula` determines the 'response' variable. This should be a single name, which should correspond to a column in `data`.

The right hand side of `formula` determines the spatial trend of the model. It specifies the linear predictor, and effectively represents the **logarithm** of the spatial trend. Variables in the formula must be the names of columns of `data`, or one of the reserved names

x,y Cartesian coordinates of location

marks Mark attached to point

id which is a factor representing the serial number (1 to n) of the point pattern, i.e. the row number in the data hyperframe.

The column of responses in `data` must consist of point patterns (objects of class `"ppp"`). The individual point pattern responses can be defined in different spatial windows. If some of the point patterns are marked, then they must all be marked, and must have the same type of marks.

The scope of models that can be fitted to each pattern is the same as the scope of `ppm`, that is, Gibbs point processes with interaction terms that belong to a specified list, including for example the Poisson process, Strauss process, Geyer's saturation model, and piecewise constant pairwise interaction models. Additionally, it is possible to include random effects as explained in the section on Random Effects below.

The stochastic part of the model is determined by the arguments `interaction` and (optionally) `iformula`.

- In the simplest case, `interaction` is an object of class `"interact"`, determining the interpoint interaction structure of the point process model, for all experimental units.
- Alternatively, `interaction` may be a hyperframe, whose entries are objects of class `"interact"`. It should have the same number of rows as `data`.
  - If `interaction` consists of only one column, then the entry in row `i` is taken to be the interpoint interaction for the `i`th experimental unit (corresponding to the `i`th row of `data`).
  - If `interaction` has more than one column, then the argument `iformula` is also required. Each row of `interaction` determines several interpoint interaction structures that might be applied to the corresponding row of `data`. The choice of interaction is determined

by `iformula`; this should be an R formula, without a left hand side. For example if `interaction` has two columns called `A` and `B` then `iformula = ~B` indicates that the interpoint interactions are taken from the second column.

Variables in `iformula` typically refer to column names of `interaction`. They can also be names of columns in `data`, but only for columns of numeric, logical or factor values. For example `iformula = ~B * group` (where `group` is a column of `data` that contains a factor) causes the model with interpoint interaction `B` to be fitted with different interaction parameters for each level of `group`.

Value

An object of class `"mppm"` representing the fitted model.

There are methods for `print`, `summary`, `coef`, `AIC`, `anova`, `fitted`, `fixef`, `logLik`, `plot`, `predict`, `ranef`, `residuals`, `summary`, `terms` and `vcov` for this class.

The default methods for `update` and `formula` also work on this class.

Random Effects

It is also possible to include random effects in the trend term. The argument `random` is a formula, with no left-hand side, that specifies the structure of the random effects. The formula should be recognisable to `lme` (see the description of the argument `random` for `lme`).

The names in the formula `random` may be any of the covariates supplied by `data`. Additionally the formula may involve the name `id`, which is a factor representing the serial number (1 to n) of the point pattern in the list `X`.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented in **spatstat** by Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<r.turner@auckland.ac.nz>` and Ege Rubak `<rubak@math.aau.dk>`.

References

Baddeley, A. and Turner, R. Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** (2000) 283–322.

Baddeley, A., Bischof, L., Sintorn, I.-M., Haggarty, S., Bell, M. and Turner, R. Analysis of a designed experiment where the response is a spatial point pattern. In preparation.

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

Bell, M. and Grunwald, G. (2004) Mixed models for the analysis of replicated spatial point patterns. *Biostatistics* **5**, 633–648.

See Also

`ppm`, `print.mppm`, `summary.mppm`, `coef.mppm`,

Examples

```
# Waterstriders data
 H <- hyperframe(Y = waterstriders)
 mppm(Y ~ 1,  data=H)
 mppm(Y ~ 1,  data=H, Strauss(7))
 mppm(Y ~ id, data=H)
 mppm(Y ~ x,  data=H)

# Synthetic data from known model
n <- 10
H <- hyperframe(V=1:n,
                U=runif(n, min=-1, max=1),
                M=factor(letters[1 + (1:n) %% 3]))
H$Z <- setcov(square(1))
H$U <- with(H, as.im(U, as.rectangle(Z)))
H$Y <- with(H, rpoispp(eval.im(exp(2+3*Z))))

fit <- mppm(Y ~Z + U + V, data=H)
```

`msr` *Signed or Vector-Valued Measure*

Description

Defines an object representing a signed measure or vector-valued measure on a spatial domain.

Usage

```
msr(qscheme, discrete, density, check=TRUE)
```

Arguments

`qscheme`	A quadrature scheme (object of class `"quad"` usually extracted from a fitted point process model).
`discrete`	Vector or matrix containing the values (masses) of the discrete component of the measure, for each of the data points in `qscheme`.
`density`	Vector or matrix containing values of the density of the diffuse component of the measure, for each of the quadrature points in `qscheme`.
`check`	Logical. Whether to check validity of the arguments.

Details

This function creates an object that represents a signed or vector valued *measure* on the two-dimensional plane. It is not normally called directly by the user.

A signed measure is a classical mathematical object (Diestel and Uhl, 1977) which can be visualised as a collection of electric charges, positive and/or negative, spread over the plane. Electric charges may be concentrated at specific points (atoms), or spread diffusely over a region.

An object of class `"msr"` represents a signed (i.e. real-valued) or vector-valued measure in the **spatstat** package.

Spatial residuals for point process models (Baddeley et al, 2005, 2008) take the form of a real-valued or vector-valued measure. The function `residuals.ppm` returns an object of class "msr" representing the residual measure.

The function `msr` would not normally be called directly by the user. It is the low-level creator function that makes an object of class "msr" from raw data.

The first argument `qscheme` is a quadrature scheme (object of class "quad"). It is typically created by `quadscheme` or extracted from a fitted point process model using `quad.ppm`. A quadrature scheme contains both data points and dummy points. The data points of `qscheme` are used as the locations of the atoms of the measure. All quadrature points (i.e. both data points and dummy points) of `qscheme` are used as sampling points for the density of the continuous component of the measure.

The argument `discrete` gives the values of the atomic component of the measure for each *data point* in `qscheme`. It should be either a numeric vector with one entry for each data point, or a numeric matrix with one row for each data point.

The argument `density` gives the values of the *density* of the diffuse component of the measure, at each *quadrature point* in `qscheme`. It should be either a numeric vector with one entry for each quadrature point, or a numeric matrix with one row for each quadrature point.

If both `discrete` and `density` are vectors (or one-column matrices) then the result is a signed (real-valued) measure. Otherwise, the result is a vector-valued measure, with the dimension of the vector space being determined by the number of columns in the matrices `discrete` and/or `density`. (If one of these is a k-column matrix and the other is a 1-column matrix, then the latter is replicated to k columns).

The class "msr" has methods for `print`, `plot` and `[`. There is also a function `Smooth.msr` for smoothing a measure.

Value

An object of class "msr" that can be plotted by `plot.msr`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

Baddeley, A., Møller, J. and Pakes, A.G. (2008) Properties of residuals for spatial point processes. *Annals of the Institute of Statistical Mathematics* **60**, 627–649.

Diestel, J. and Uhl, J.J. Jr (1977) *Vector measures*. Providence, RI, USA: American Mathematical Society.

Halmos, P.R. (1950) *Measure Theory*. Van Nostrand.

See Also

`plot.msr`, `Smooth.msr`, `[.msr`, `with.msr`, `split.msr`, `Ops.msr`, `measureVariation`.

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)

rp <- residuals(fit, type="pearson")
rp

rs <- residuals(fit, type="score")
rs
colnames(rs)

# An equivalent way to construct the Pearson residual measure by hand
Q <- quad.ppm(fit)
lambda <- fitted(fit)
slam <- sqrt(lambda)
Z <- is.data(Q)
m <- msr(Q, discrete=1/slam[Z], density = -slam)
m
```

MultiHard *The Multitype Hard Core Point Process Model*

Description

Creates an instance of the multitype hard core point process model which can then be fitted to point pattern data.

Usage

```
MultiHard(hradii, types=NULL)
```

Arguments

hradii	Matrix of hard core radii
types	Optional; vector of all possible types (i.e. the possible levels of the marks variable in the data)

Details

This is a multitype version of the hard core process. A pair of points of types i and j must not lie closer than h_{ij} units apart.

The argument types need not be specified in normal use. It will be determined automatically from the point pattern data set to which the MultiStrauss interaction is applied, when the user calls ppm. However, the user should be confident that the ordering of types in the dataset corresponds to the ordering of rows and columns in the matrix hradii.

The matrix hradii must be symmetric, with entries which are either positive numbers or NA. A value of NA indicates that no distance constraint should be applied for this combination of types.

Note that only the hardcore radii are specified in MultiHard. The canonical parameters $\log(\beta_j)$ are estimated by ppm(), not fixed in MultiHard().

Value

An object of class "interact" describing the interpoint interaction structure of the multitype hard core process with hard core radii $hradii[i, j]$.

Warnings

In order that ppm can fit the multitype hard core model correctly to a point pattern X, this pattern must be marked, with markformat equal to vector and the mark vector marks(X) must be a factor. If the argument types is specified it is interpreted as a set of factor levels and this set must equal levels(marks(X)).

Changed Syntax

Before **spatstat** version 1.37-0, the syntax of this function was different: MultiHard(types=NULL, hradii). The new code attempts to handle the old syntax as well.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

ppm, pairwise.family, ppm.object, MultiStrauss, MultiStraussHard, Strauss.

See ragsMultiHard and rmh for simulation.

Examples

```
h <- matrix(c(1,2,2,1), nrow=2,ncol=2)

# prints a sensible description of itself
MultiHard(h)

# Fit the stationary multitype hardcore process to `amacrine'
# with hard core operating only between cells of the same type.
h <- 0.02 * matrix(c(1, NA, NA, 1), nrow=2,ncol=2)
ppm(amacrine ~1, MultiHard(h))
```

---

multiplicity.ppp *Count Multiplicity of Duplicate Points*

Description

Counts the number of duplicates for each point in a spatial point pattern.

Usage

```
multiplicity(x)

## S3 method for class 'ppp'
multiplicity(x)

## S3 method for class 'ppx'
multiplicity(x)

## S3 method for class 'data.frame'
multiplicity(x)

## Default S3 method:
multiplicity(x)
```

Arguments

x	A spatial point pattern (object of class "ppp" or "ppx") or a vector, matrix or data frame.

Details

Two points in a point pattern are deemed to be identical if their x, y coordinates are the same, and their marks are also the same (if they carry marks). The Examples section illustrates how it is possible for a point pattern to contain a pair of identical points.

For each point in x, the function multiplicity counts how many points are identical to it, and returns the vector of counts.

The argument x can also be a vector, a matrix or a data frame. When x is a vector, m <- multiplicity(x) is a vector of the same length as x, and m[i] is the number of elements of x that are identical to x[i]. When x is a matrix or data frame, m <- multiplicity(x) is a vector of length equal to the number of rows of x, and m[i] is the number of rows of x that are identical to the ith row.

Value

A vector of integers (multiplicities) of length equal to the number of points in x.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Sebastian Meyer.

See Also

ppp.object, duplicated.ppp, unique.ppp

Examples

```
X <- ppp(c(1,1,0.5,1), c(2,2,1,2), window=square(3), check=FALSE)
m <- multiplicity(X)

# unique points in X, marked by their multiplicity
```

```
first <- !duplicated(X)
Y <- X[first] %mark% m[first]
```

MultiStrauss *The Multitype Strauss Point Process Model*

Description

Creates an instance of the multitype Strauss point process model which can then be fitted to point pattern data.

Usage

```
MultiStrauss(radii, types=NULL)
```

Arguments

`radii`	Matrix of interaction radii
`types`	Optional; vector of all possible types (i.e. the possible levels of the `marks` variable in the data)

Details

The (stationary) multitype Strauss process with m types, with interaction radii r_{ij} and parameters β_j and γ_{ij} is the pairwise interaction point process in which each point of type j contributes a factor β_j to the probability density of the point pattern, and a pair of points of types i and j closer than r_{ij} units apart contributes a factor γ_{ij} to the density.

The nonstationary multitype Strauss process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location and type, rather than a constant beta.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the multitype Strauss process pairwise interaction is yielded by the function `MultiStrauss()`. See the examples below.

The argument `types` need not be specified in normal use. It will be determined automatically from the point pattern data set to which the MultiStrauss interaction is applied, when the user calls `ppm`. However, the user should be confident that the ordering of types in the dataset corresponds to the ordering of rows and columns in the matrix `radii`.

The matrix `radii` must be symmetric, with entries which are either positive numbers or `NA`. A value of `NA` indicates that no interaction term should be included for this combination of types.

Note that only the interaction radii are specified in `MultiStrauss`. The canonical parameters $\log(\beta_j)$ and $\log(\gamma_{ij})$ are estimated by `ppm()`, not fixed in `MultiStrauss()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the multitype Strauss process with interaction radii $radii[i,j]$.

Warnings

In order that ppm can fit the multitype Strauss model correctly to a point pattern X, this pattern must be marked, with markformat equal to vector and the mark vector marks(X) must be a factor. If the argument types is specified it is interpreted as a set of factor levels and this set must equal levels(marks(X)).

Changed Syntax

Before **spatstat** version 1.37-0, the syntax of this function was different: MultiStrauss(types=NULL, radii). The new code attempts to handle the old syntax as well.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

ppm, pairwise.family, ppm.object, Strauss, MultiHard

Examples

```
   r <- matrix(c(1,2,2,1), nrow=2,ncol=2)
   MultiStrauss(r)
   # prints a sensible description of itself
   r <- 0.03 * matrix(c(1,2,2,1), nrow=2,ncol=2)
   X <- amacrine

   ppm(X ~1, MultiStrauss(r))
   # fit the stationary multitype Strauss process to `amacrine'

   ## Not run:
   ppm(X ~polynom(x,y,3), MultiStrauss(r, c("off","on")))
   # fit a nonstationary multitype Strauss process with log-cubic trend

## End(Not run)
```

MultiStraussHard   *The Multitype/Hard Core Strauss Point Process Model*

Description

Creates an instance of the multitype/hard core Strauss point process model which can then be fitted to point pattern data.

Usage

```
MultiStraussHard(iradii, hradii, types=NULL)
```

Arguments

`iradii`	Matrix of interaction radii
`hradii`	Matrix of hard core radii
`types`	Optional; vector of all possible types (i.e. the possible levels of the `marks` variable in the data)

Details

This is a hybrid of the multitype Strauss process (see `MultiStrauss`) and the hard core process (case $\gamma = 0$ of the Strauss process). A pair of points of types i and j must not lie closer than h_{ij} units apart; if the pair lies more than h_{ij} and less than r_{ij} units apart, it contributes a factor γ_{ij} to the probability density.

The argument `types` need not be specified in normal use. It will be determined automatically from the point pattern data set to which the MultiStraussHard interaction is applied, when the user calls `ppm`. However, the user should be confident that the ordering of types in the dataset corresponds to the ordering of rows and columns in the matrices `iradii` and `hradii`.

The matrices `iradii` and `hradii` must be symmetric, with entries which are either positive numbers or `NA`. A value of `NA` indicates that no interaction term should be included for this combination of types.

Note that only the interaction radii and hardcore radii are specified in `MultiStraussHard`. The canonical parameters $\log(\beta_j)$ and $\log(\gamma_{ij})$ are estimated by `ppm()`, not fixed in `MultiStraussHard()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the multitype/hard core Strauss process with interaction radii $iradii[i, j]$ and hard core radii $hradii[i, j]$.

Warnings

In order that `ppm` can fit the multitype/hard core Strauss model correctly to a point pattern `X`, this pattern must be marked, with `markformat` equal to `vector` and the mark vector `marks(X)` must be a factor. If the argument `types` is specified it is interpreted as a set of factor levels and this set must equal `levels(marks(X))`.

Changed Syntax

Before **spatstat** version `1.37-0`, the syntax of this function was different: `MultiStraussHard(types=NULL, iradii, h`
The new code attempts to handle the old syntax as well.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

, Rolf Turner `<r.turner@auckland.ac.nz>`

and Ege Rubak `<rubak@math.aau.dk>`

See Also

`ppm`, `pairwise.family`, `ppm.object`, `MultiStrauss`, `MultiHard`, `Strauss`

Examples

```
r <- matrix(3, nrow=2,ncol=2)
h <- matrix(c(1,2,2,1), nrow=2,ncol=2)
MultiStraussHard(r,h)
# prints a sensible description of itself
r <- 0.04 * matrix(c(1,2,2,1), nrow=2,ncol=2)
h <- 0.02 * matrix(c(1,NA,NA,1), nrow=2,ncol=2)
X <- amacrine

fit <- ppm(X ~1, MultiStraussHard(r,h))
# fit stationary multitype hardcore Strauss process to `amacrine'
```

`nearest.raster.point` *Find Pixel Nearest to a Given Point*

Description

Given cartesian coordinates, find the nearest pixel.

Usage

```
nearest.raster.point(x,y,w, indices=TRUE)
```

Arguments

`x`	Numeric vector of x coordinates of any points
`y`	Numeric vector of y coordinates of any points
`w`	An image (object of class `"im"`) or a binary mask window (an object of class `"owin"` of type `"mask"`).
`indices`	Logical flag indicating whether to return the row and column indices, or the actual x, y coordinates.

Details

The argument `w` should be either a pixel image (object of class `"im"`) or a window (an object of class `"owin"`, see `owin.object` for details) of type `"mask"`.

The arguments `x` and `y` should be numeric vectors of equal length. They are interpreted as the coordinates of points in space. For each point `(x[i], y[i])`, the function finds the nearest pixel in the grid of pixels for `w`.

If `indices=TRUE`, this function returns a list containing two vectors `rr` and `cc` giving row and column positions (in the image matrix). For the location `(x[i],y[i])` the nearest pixel is at row `rr[i]` and column `cc[i]` of the image.

If `indices=FALSE`, the function returns a list containing two vectors `x` and `y` giving the actual coordinates of the pixels.

Value

If `indices=TRUE`, a list containing two vectors `rr` and `cc` giving row and column positions (in the image matrix). If `indices=FALSE`, a list containing vectors `x` and `y` giving actual coordinates of the pixels.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin.object, as.mask

Examples

```
w <- owin(c(0,1), c(0,1), mask=matrix(TRUE, 100,100))  # 100 x 100 grid
nearest.raster.point(0.5, 0.3, w)
nearest.raster.point(0.5, 0.3, w, indices=FALSE)
```

---

nearestsegment *Find Line Segment Nearest to Each Point*

Description

Given a point pattern and a line segment pattern, this function finds the nearest line segment for each point.

Usage

```
nearestsegment(X, Y)
```

Arguments

X	A point pattern (object of class "ppp").
Y	A line segment pattern (object of class "psp").

Details

The distance between a point x and a straight line segment y is defined to be the shortest Euclidean distance between x and any location on y. This algorithm first calculates the distance from each point of X to each segment of Y. Then it determines, for each point x in X, which segment of Y is closest. The index of this segment is returned.

Value

Integer vector v (of length equal to the number of points in X) identifying the nearest segment to each point. If v[i] = j, then Y[j] is the line segment lying closest to X[i].

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

project2segment to project each point of X to a point lying on one of the line segments.

Use distmap.psp to identify the nearest line segment for each pixel in a grid.

Examples

```
X <- runifpoint(3)
Y <- as.psp(matrix(runif(20), 5, 4), window=owin())
v <- nearestsegment(X,Y)
plot(Y)
plot(X, add=TRUE)
plot(X[1], add=TRUE, col="red")
plot(Y[v[1]], add=TRUE, lwd=2, col="red")
```

---

nestsplit *Nested Split*

Description

Applies two splitting operations to a point pattern, producing a list of lists of patterns.

Usage

```
nestsplit(X, ...)
```

Arguments

X	Point pattern to be split. Object of class "ppp".
...	Data determining the splitting factors or splitting regions. See Details.

Details

This function splits the point pattern X into several sub-patterns using split.ppp, then splits each of the sub-patterns into sub-sub-patterns using split.ppp again. The result is a hyperframe containing the sub-sub-patterns and two factors indicating the grouping.

The arguments ... determine the two splitting factors or splitting regions. Each argument may be:

- a factor (of length equal to the number of points in X)
- the name of a column of marks of X (provided this column contains factor values)
- a tessellation (class "tess")
- a pixel image (class "im") with factor values
- a window (class "owin")
- identified by name (in the form name=value) as one of the formal arguments of quadrats or tess

The arguments will be processed to yield a list of two splitting factors/tessellations. The splits will be applied to X consecutively to produce the sub-sub-patterns.

Value

A hyperframe with three columns. The first column contains the sub-sub-patterns. The second and third columns are factors which identify the grouping according to the two splitting factors.

Author(s)

Original idea by Ute Hahn. Code by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

split.ppp, quantess

Examples

```
# factor and tessellation
Nft <- nestsplit(amacrine, marks(amacrine), quadrats(amacrine, 3, 1))
Ntf <- nestsplit(amacrine, quadrats(amacrine, 3, 1), marks(amacrine))
Ntf

# two factors
big <- with(marks(betacells), area > 300)
Nff <- nestsplit(betacells, "type", factor(big))

# two tessellations
Tx <- quantess(redwood, "x", 4)
Td <- dirichlet(runifpoint(5, Window(redwood)))
Ntt <- nestsplit(redwood, Td, Tx)
Ntt2 <- nestsplit(redwood, Td, ny=3)
```

---

nnclean *Nearest Neighbour Clutter Removal*

Description

Detect features in a 2D or 3D spatial point pattern using nearest neighbour clutter removal.

Usage

```
nnclean(X, k, ...)

## S3 method for class 'ppp'
nnclean(X, k, ...,
                     edge.correct = FALSE, wrap = 0.1,
                     convergence = 0.001, plothist = FALSE,
                     verbose = TRUE, maxit = 50)

## S3 method for class 'pp3'
nnclean(X, k, ...,
                     convergence = 0.001, plothist = FALSE,
                     verbose = TRUE, maxit = 50)
```

Arguments

`X`	A two-dimensional spatial point pattern (object of class `"ppp"`) or a three-dimensional point pattern (object of class `"pp3"`).
`k`	Degree of neighbour: `k=1` means nearest neighbour, `k=2` means second nearest, etc.
`...`	Arguments passed to `hist.default` to control the appearance of the histogram, if `plothist=TRUE`.
`edge.correct`	Logical flag specifying whether periodic edge correction should be performed (only implemented in 2 dimensions).
`wrap`	Numeric value specifying the relative size of the margin in which data will be replicated for the periodic edge correction (if `edge.correct=TRUE`). A fraction of window width and window height.
`convergence`	Relative tolerance threshold for testing convergence of EM algorithm.
`maxit`	Maximum number of iterations for EM algorithm.
`plothist`	Logical flag specifying whether to plot a diagnostic histogram of the nearest neighbour distances and the fitted distribution.
`verbose`	Logical flag specifying whether to print progress reports.

Details

Byers and Raftery (1998) developed a technique for recognising features in a spatial point pattern in the presence of random clutter.

For each point in the pattern, the distance to the kth nearest neighbour is computed. Then the E-M algorithm is used to fit a mixture distribution to the kth nearest neighbour distances. The mixture components represent the feature and the clutter. The mixture model can be used to classify each point as belong to one or other component.

The function `nnclean` is generic, with methods for two-dimensional point patterns (class `"ppp"`) and three-dimensional point patterns (class `"pp3"`) currently implemented.

The result is a point pattern (2D or 3D) with two additional columns of marks:

class A factor, with levels `"noise"` and `"feature"`, indicating the maximum likelihood classification of each point.

prob Numeric vector giving the estimated probabilities that each point belongs to a feature.

The object also has extra information stored in attributes: `"theta"` contains the fitted parameters of the mixture model, `"info"` contains information about the fitting procedure, and `"hist"` contains the histogram structure returned from `hist.default` if `plothist = TRUE`.

Value

An object of the same kind as `X`, obtained by attaching marks to the points of `X`.

The object also has attributes, as described under Details.

Author(s)

Original by Simon Byers and Adrian Raftery. Adapted for **spatstat** by Adrian Baddeley `<Adrian.Baddeley@curtin.edu`

References

Byers, S. and Raftery, A.E. (1998) Nearest-neighbour clutter removal for estimating features in spatial point processes. *Journal of the American Statistical Association* **93**, 577–584.

See Also

nndist, split.ppp, cut.ppp

Examples

```
data(shapley)
X <- nnclean(shapley, k=17, plothist=TRUE)
plot(X, which.marks=1, chars=c(".", "+"), cols=1:2)
plot(X, which.marks=2, cols=function(x)hsv(0.2+0.8*(1-x),1,1))
Y <- split(X, un=TRUE)
plot(Y, chars="+", cex=0.5)
marks(X) <- marks(X)$prob
plot(cut(X, breaks=3), chars=c(".", "+", "+"), cols=1:3)
```

---

nncorr *Nearest-Neighbour Correlation Indices of Marked Point Pattern*

Description

Computes nearest-neighbour correlation indices of a marked point pattern, including the nearest-neighbour mark product index (default case of nncorr), the nearest-neighbour mark index (nnmean), and the nearest-neighbour variogram index (nnvario).

Usage

```
nncorr(X,
       f = function(m1, m2) { m1 * m2 },
       k = 1,
       ...,
       use = "all.obs", method = c("pearson", "kendall", "spearman"),
       denominator=NULL)
nnmean(X, k=1)
nnvario(X, k=1)
```

Arguments

X	The observed point pattern. An object of class "ppp".
f	Function f used in the definition of the nearest neighbour correlation. There is a sensible default that depends on the type of marks of X.
k	Integer. The k-th nearest neighbour of each point will be used.
...	Extra arguments passed to f.
use,method	Arguments passed to the standard correlation function cor.
denominator	Internal use only.

Details

The nearest neighbour correlation index $\bar{n}_f$ of a marked point process X is a number measuring the dependence between the mark of a typical point and the mark of its nearest neighbour.

The command `nncorr` computes the nearest neighbour correlation index based on any test function `f` provided by the user. The default behaviour of `nncorr` is to compute the nearest neighbour mark product index. The commands `nnmean` and `nnvario` are convenient abbreviations for other special choices of `f`.

In the default case, `nncorr(X)` computes three different versions of the nearest-neighbour correlation index: the unnormalised, normalised, and classical correlations.

unnormalised: The **unnormalised** nearest neighbour correlation (Stoyan and Stoyan, 1994, section 14.7) is defined as

$$\bar{n}_f = E[f(M, M^*)]$$

where $E[]$ denotes mean value, M is the mark attached to a typical point of the point process, and M^* is the mark attached to its nearest neighbour (i.e. the nearest other point of the point process).

Here f is any function $f(m_1, m_2)$ with two arguments which are possible marks of the pattern, and which returns a nonnegative real value. Common choices of f are: for continuous real-valued marks,

$$f(m_1, m_2) = m_1 m_2$$

for discrete marks (multitype point patterns),

$$f(m_1, m_2) = 1(m_1 = m_2)$$

and for marks taking values in $[0, 2\pi)$,

$$f(m_1, m_2) = \sin(m_1 - m_2)$$

For example, in the second case, the unnormalised nearest neighbour correlation $\bar{n}_f$ equals the proportion of points in the pattern which have the same mark as their nearest neighbour.

Note that $\bar{n}_f$ is not a "correlation" in the usual statistical sense. It can take values greater than 1.

normalised: We can define a **normalised** nearest neighbour correlation by

$$\bar{m}_f = \frac{E[f(M, M^*)]}{E[f(M, M')]}$$

where again M is the mark attached to a typical point, M^* is the mark attached to its nearest neighbour, and M' is an independent copy of M with the same distribution. This normalisation is also not a "correlation" in the usual statistical sense, but is normalised so that the value 1 suggests "lack of correlation": if the marks attached to the points of `X` are independent and identically distributed, then $\bar{m}_f = 1$. The interpretation of values larger or smaller than 1 depends on the choice of function f.

classical: Finally if the marks of `X` are real numbers, we can also compute the **classical** correlation, that is, the correlation coefficient of the two random variables M and M^*. The classical correlation has a value between -1 and 1. Values close to -1 or 1 indicate strong dependence between the marks.

In the default case where `f` is not given, `nncorr(X)` computes

- If the marks of `X` are real numbers, the unnormalised and normalised versions of the nearest-neighbour product index $E[M\, M^*]$, and the classical correlation between M and M^*.

- If the marks of X are factor valued, the unnormalised and normalised versions of the nearest-neighbour equality index $P[M = M^*]$.

The wrapper functions nnmean and nnvario compute the correlation indices for two special choices of the function $f(m_1, m_2)$.

- nnmean computes the correlation indices for $f(m_1, m_2) = m_1$. The unnormalised index is simply the mean value of the mark of the neighbour of a typical point, $E[M^*]$, while the normalised index is $E[M^*]/E[M]$, the ratio of the mean mark of the neighbour of a typical point to the mean mark of a typical point.
- nnvario computes the correlation indices for $f(m_1, m_2) = (1/2)(m_1 - m_2)^2$.

The argument X must be a point pattern (object of class "ppp") and must be a marked point pattern. (The marks may be a data frame, containing several columns of mark variables; each column is treated separately.)

If the argument f is given, it must be a function, accepting two arguments m1 and m2 which are vectors of equal length containing mark values (of the same type as the marks of X). It must return a vector of numeric values of the same length as m1 and m2. The values must be non-negative.

The arguments use and method control the calculation of the classical correlation using cor, as explained in the help file for cor.

Other arguments may be passed to f through the ... argument.

This algorithm assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape. Biases due to edge effects are treated using the 'border method' edge correction.

Value

Labelled vector of length 2 or 3 containing the unnormalised and normalised nearest neighbour correlations, and the classical correlation if appropriate. Alternatively a matrix with 2 or 3 rows, containing this information for each mark variable.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

Examples

```
data(finpines)
nncorr(finpines)
# heights of neighbouring trees are slightly negatively correlated

data(amacrine)
nncorr(amacrine)
# neighbouring cells are usually of different type
```

`nncross` *Nearest Neighbours Between Two Patterns*

Description

Given two point patterns X and Y, finds the nearest neighbour in Y of each point of X. Alternatively Y may be a line segment pattern.

Usage

```
  nncross(X, Y, ...)

  ## S3 method for class 'ppp'
nncross(X, Y,
          iX=NULL, iY=NULL,
          what = c("dist", "which"),
          ...,
          k = 1,
          sortby=c("range", "var", "x", "y"),
          is.sorted.X = FALSE,
          is.sorted.Y = FALSE)

  ## Default S3 method:
nncross(X, Y, ...)
```

Arguments

X	Point pattern (object of class "ppp").
Y	Either a point pattern (object of class "ppp") or a line segment pattern (object of class "psp").
iX, iY	Optional identifiers, applicable only in the case where Y is a point pattern, used to determine whether a point in X is identical to a point in Y. See Details.
what	Character string specifying what information should be returned. Either the nearest neighbour distance ("dist"), the identifier of the nearest neighbour ("which"), or both.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.
sortby	Determines which coordinate to use to sort the point patterns. See Details.
is.sorted.X, is.sorted.Y	Logical values attesting whether the point patterns X and Y have been sorted. See Details.
...	Ignored.

Details

Given two point patterns X and Y this function finds, for each point of X, the nearest point of Y. The distance between these points is also computed. If the argument k is specified, then the k-th nearest neighbours will be found.

Alternatively if X is a point pattern and Y is a line segment pattern, the function finds the nearest line segment to each point of X, and computes the distance.

The return value is a data frame, with rows corresponding to the points of X. The first column gives the nearest neighbour distances (i.e. the ith entry is the distance from the ith point of X to the nearest element of Y). The second column gives the indices of the nearest neighbours (i.e.\ the ith entry is the index of the nearest element in Y.) If what="dist" then only the vector of distances is returned. If what="which" then only the vector of indices is returned.

The argument k may be an integer or an integer vector. If it is a single integer, then the k-th nearest neighbours are computed. If it is a vector, then the k[i]-th nearest neighbours are computed for each entry k[i]. For example, setting k=1:3 will compute the nearest, second-nearest and third-nearest neighbours. The result is a data frame.

Note that this function is not symmetric in X and Y. To find the nearest neighbour in X of each point in Y, where Y is a point pattern, use nncross(Y,X).

The arguments iX and iY are used when the two point patterns X and Y have some points in common. In this situation nncross(X, Y) would return some zero distances. To avoid this, attach a unique integer identifier to each point, such that two points are identical if their identifying numbers are equal. Let iX be the vector of identifier values for the points in X, and iY the vector of identifiers for points in Y. Then the code will only compare two points if they have different values of the identifier. See the Examples.

Value

A data frame, or a vector if the data frame would contain only one column.

By default (if what=c("dist", "which") and k=1) a data frame with two columns:

dist	Nearest neighbour distance
which	Nearest neighbour index in Y

If what="dist" and k=1, a vector of nearest neighbour distances.

If what="which" and k=1, a vector of nearest neighbour indices.

If k is specified, the result is a data frame with columns containing the k-th nearest neighbour distances and/or nearest neighbour indices.

Sorting data and pre-sorted data

Read this section if you care about the speed of computation.

For efficiency, the algorithm sorts the point patterns X and Y into increasing order of the x coordinate or increasing order of the the y coordinate. Sorting is only an intermediate step; it does not affect the output, which is always given in the same order as the original data.

By default (if sortby="range"), the sorting will occur on the coordinate that has the larger range of values (according to the frame of the enclosing window of Y). If sortby = "var"), sorting will occur on the coordinate that has the greater variance (in the pattern Y). Setting sortby="x" or sortby = "y" will specify that sorting should occur on the x or y coordinate, respectively.

If the point pattern X is already sorted, then the corresponding argument is.sorted.X should be set to TRUE, and sortby should be set equal to "x" or "y" to indicate which coordinate is sorted.

Similarly if Y is already sorted, then is.sorted.Y should be set to TRUE, and sortby should be set equal to "x" or "y" to indicate which coordinate is sorted.

If both X and Y are sorted *on the same coordinate axis* then both is.sorted.X and is.sorted.Y should be set to TRUE, and sortby should be set equal to "x" or "y" to indicate which coordinate is sorted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz> , and Jens Oehlschlaegel

See Also

nndist for nearest neighbour distances in a single point pattern.

Examples

```
# two different point patterns
X <- runifpoint(15)
Y <- runifpoint(20)
N <- nncross(X,Y)$which
# note that length(N) = 15
plot(superimpose(X=X,Y=Y), main="nncross", cols=c("red","blue"))
arrows(X$x, X$y, Y[N]$x, Y[N]$y, length=0.15)

# third-nearest neighbour
NXY <- nncross(X, Y, k=3)
NXY[1:3,]
# second and third nearest neighbours
NXY <- nncross(X, Y, k=2:3)
NXY[1:3,]

# two patterns with some points in common
Z <- runifpoint(50)
X <- Z[1:30]
Y <- Z[20:50]
iX <- 1:30
iY <- 20:50
N <- nncross(X,Y, iX, iY)$which
N <- nncross(X,Y, iX, iY, what="which") #faster
plot(superimpose(X=X, Y=Y), main="nncross", cols=c("red","blue"))
arrows(X$x, X$y, Y[N]$x, Y[N]$y, length=0.15)

# point pattern and line segment pattern
X <- runifpoint(15)
Y <- rpoisline(10)
N <- nncross(X,Y)
```

---

nncross.lpp *Nearest Neighbours on a Linear Network*

Description

Given two point patterns X and Y on a linear network, finds the nearest neighbour in Y of each point of X using the shortest path in the network.

Usage

```
  ## S3 method for class 'lpp'
nncross(X, Y,
          iX=NULL, iY=NULL,
          what = c("dist", "which"),
          ...,
          k = 1,
          method="C")
```

Arguments

`X,Y`	Point patterns on a linear network (objects of class `"lpp"`). They must lie on the *same* linear network.
`iX, iY`	Optional identifiers, used to determine whether a point in `X` is identical to a point in `Y`. See Details.
`what`	Character string specifying what information should be returned. Either the nearest neighbour distance (`"dist"`), the identifier of the nearest neighbour (`"which"`), or both.
`...`	Ignored.
`k`	Integer, or integer vector. The algorithm will compute the distance to the `k`th nearest neighbour, for each value of `k`.
`method`	Internal use only.

Details

Given two point patterns `X` and `Y` on the same linear network, this function finds, for each point of `X`, the nearest point of `Y`, measuring distance by the shortest path in the network. The distance between these points is also computed.

The return value is a data frame, with rows corresponding to the points of `X`. The first column gives the nearest neighbour distances (i.e. the `i`th entry is the distance from the `i`th point of `X` to the nearest element of `Y`). The second column gives the indices of the nearest neighbours (i.e.\ the `i`th entry is the index of the nearest element in `Y`.) If `what="dist"` then only the vector of distances is returned. If `what="which"` then only the vector of indices is returned.

Note that this function is not symmetric in `X` and `Y`. To find the nearest neighbour in `X` of each point in `Y`, use `nncross(Y,X)`.

The arguments `iX` and `iY` are used when the two point patterns `X` and `Y` have some points in common. In this situation `nncross(X, Y)` would return some zero distances. To avoid this, attach a unique integer identifier to each point, such that two points are identical if their identifying numbers are equal. Let `iX` be the vector of identifier values for the points in `X`, and `iY` the vector of identifiers for points in `Y`. Then the code will only compare two points if they have different values of the identifier. See the Examples.

The `k`th nearest neighbour may be undefined, for example if there are fewer than `k+1` points in the dataset, or if the linear network is not connected. In this case, the `k`th nearest neighbour distance is infinite.

Value

By default (if `what=c("dist", "which")` and `k=1`) a data frame with two columns:

`dist`	Nearest neighbour distance

which Nearest neighbour index in Y

If what="dist", a vector of nearest neighbour distances.

If what="which", a vector of nearest neighbour indices.

If k is a vector of integers, the result is a matrix with one row for each point in X, giving the distances and/or indices of the kth nearest neighbours in Y.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

nndist.lpp for nearest neighbour distances in a single point pattern.

nnwhich.lpp to identify which points are nearest neighbours in a single point pattern.

Examples

```
# two different point patterns
X <- runiflpp(3, simplenet)
Y <- runiflpp(5, simplenet)
nn <- nncross(X,Y)
nn
plot(simplenet, main="nncross")
plot(X, add=TRUE, cols="red")
plot(Y, add=TRUE, cols="blue", pch=16)
XX <- as.ppp(X)
YY <- as.ppp(Y)
i <- nn$which
arrows(XX$x, XX$y, YY[i]$x, YY[i]$y, length=0.15)

# nearest and second-nearest neighbours
nncross(X, Y, k=1:2)

# two patterns with some points in common
X <- Y[1:2]
iX <- 1:2
iY <- 1:5
nncross(X,Y, iX, iY)
```

---

nncross.pp3 *Nearest Neighbours Between Two Patterns in 3D*

Description

Given two point patterns X and Y in three dimensions, finds the nearest neighbour in Y of each point of X.

Usage

```
  ## S3 method for class 'pp3'
nncross(X, Y,
          iX=NULL, iY=NULL,
          what = c("dist", "which"),
          ...,
          k = 1,
          sortby=c("range", "var", "x", "y", "z"),
          is.sorted.X = FALSE,
          is.sorted.Y = FALSE)
```

Arguments

`X,Y`	Point patterns in three dimensions (objects of class "`pp3`").
`iX, iY`	Optional identifiers, used to determine whether a point in `X` is identical to a point in `Y`. See Details.
`what`	Character string specifying what information should be returned. Either the nearest neighbour distance ("`dist`"), the identifier of the nearest neighbour ("`which`"), or both.
`k`	Integer, or integer vector. The algorithm will compute the distance to the `k`th nearest neighbour.
`sortby`	Determines which coordinate to use to sort the point patterns. See Details.
`is.sorted.X, is.sorted.Y`	Logical values attesting whether the point patterns `X` and `Y` have been sorted. See Details.
`...`	Ignored.

Details

Given two point patterns `X` and `Y` in three dimensions, this function finds, for each point of `X`, the nearest point of `Y`. The distance between these points is also computed. If the argument `k` is specified, then the `k`-th nearest neighbours will be found.

The return value is a data frame, with rows corresponding to the points of `X`. The first column gives the nearest neighbour distances (i.e. the `i`th entry is the distance from the `i`th point of `X` to the nearest element of `Y`). The second column gives the indices of the nearest neighbours (i.e.\ the `i`th entry is the index of the nearest element in `Y`.) If `what="dist"` then only the vector of distances is returned. If `what="which"` then only the vector of indices is returned.

The argument `k` may be an integer or an integer vector. If it is a single integer, then the `k`-th nearest neighbours are computed. If it is a vector, then the `k[i]`-th nearest neighbours are computed for each entry `k[i]`. For example, setting `k=1:3` will compute the nearest, second-nearest and third-nearest neighbours. The result is a data frame.

Note that this function is not symmetric in `X` and `Y`. To find the nearest neighbour in `X` of each point in `Y`, use `nncross(Y,X)`.

The arguments `iX` and `iY` are used when the two point patterns `X` and `Y` have some points in common. In this situation `nncross(X, Y)` would return some zero distances. To avoid this, attach a unique integer identifier to each point, such that two points are identical if their identifying numbers are equal. Let `iX` be the vector of identifier values for the points in `X`, and `iY` the vector of identifiers for points in `Y`. Then the code will only compare two points if they have different values of the identifier. See the Examples.

Value

A data frame, or a vector if the data frame would contain only one column.

By default (if `what=c("dist", "which")` and `k=1`) a data frame with two columns:

`dist` Nearest neighbour distance

`which` Nearest neighbour index in `Y`

If `what="dist"` and `k=1`, a vector of nearest neighbour distances.

If `what="which"` and `k=1`, a vector of nearest neighbour indices.

If `k` is specified, the result is a data frame with columns containing the `k`-th nearest neighbour distances and/or nearest neighbour indices.

Sorting data and pre-sorted data

Read this section if you care about the speed of computation.

For efficiency, the algorithm sorts both the point patterns `X` and `Y` into increasing order of the x coordinate, or both into increasing order of the y coordinate, or both into increasing order of the z coordinate. Sorting is only an intermediate step; it does not affect the output, which is always given in the same order as the original data.

By default (if `sortby="range"`), the sorting will occur on the coordinate that has the largest range of values (according to the frame of the enclosing window of `Y`). If `sortby = "var"`), sorting will occur on the coordinate that has the greater variance (in the pattern `Y`). Setting `sortby="x"` or `sortby = "y"` or `sortby = "z"` will specify that sorting should occur on the x, y or z coordinate, respectively.

If the point pattern `X` is already sorted, then the corresponding argument `is.sorted.X` should be set to `TRUE`, and `sortby` should be set equal to `"x"`, `"y"` or `"z"` to indicate which coordinate is sorted.

Similarly if `Y` is already sorted, then `is.sorted.Y` should be set to `TRUE`, and `sortby` should be set equal to `"x"`, `"y"` or `"z"` to indicate which coordinate is sorted.

If both `X` and `Y` are sorted *on the same coordinate axis* then both `is.sorted.X` and `is.sorted.Y` should be set to `TRUE`, and `sortby` should be set equal to `"x"`, `"y"` or `"z"` to indicate which coordinate is sorted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz> , and Jens Oehlschlaegel

See Also

`nndist` for nearest neighbour distances in a single point pattern.

Examples

```
# two different point patterns
X <- pp3(runif(10), runif(10), runif(10), box3(c(0,1)))
Y <- pp3(runif(20), runif(20), runif(20), box3(c(0,1)))
N <- nncross(X,Y)$which
N <- nncross(X,Y, what="which") #faster
# note that length(N) = 10

# k-nearest neighbours
```

```
N3 <- nncross(X, Y, k=1:3)

# two patterns with some points in common
Z <- pp3(runif(20), runif(20), runif(20), box3(c(0,1)))
X <- Z[1:15]
Y <- Z[10:20]
iX <- 1:15
iY <- 10:20
N <- nncross(X,Y, iX, iY, what="which")
```

---

nndensity.ppp *Estimate Intensity of Point Pattern Using Nearest Neighbour Distances*

Description

Estimates the intensity of a point pattern using the distance from each spatial location to the kth nearest data point.

Usage

```
nndensity(x, ...)

## S3 method for class 'ppp'
nndensity(x, k, ..., verbose = TRUE)
```

Arguments

x	A point pattern (object of class "ppp") or some other spatial object.
k	Integer. The distance to the kth nearest data point will be computed. There is a sensible default.
...	Arguments passed to nnmap and as.mask controlling the pixel resolution.
verbose	Logical. If TRUE, print the value of k when it is automatically selected. If FALSE, remain silent.

Details

This function computes a quick estimate of the intensity of the point process that generated the point pattern x.

For each spatial location s, let $d(s)$ be the distance from s to the k-th nearest point in the dataset x. If the data came from a homogeneous Poisson process with intensity λ, then $\pi d(s)^2$ would follow a negative exponential distribution with mean $1/\lambda$, and the maximum likelihood estimate of λ would be $1/(\pi d(s)^2)$. This is the estimate computed by nndensity, apart from an edge effect correction.

This estimator of intensity is relatively fast to compute, and is spatially adaptive (so that it can handle wide variation in the intensity function). However, it implicitly assumes the points are independent, so it does not perform well if the pattern is strongly clustered or strongly inhibited.

The value of k should be greater than 1 in order to avoid infinite peaks in the intensity estimate around each data point. The default value of k is the square root of the number of points in x, which seems to work well in many cases.

The window of x is digitised using as.mask and the values $d(s)$ are computed using nnmap. To control the pixel resolution, see as.mask.

Value

A pixel image (object of class "im") giving the estimated intensity of the point process at each spatial location. Pixel values are intensities (number of points per unit area).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

NEED REFERENCES. TRY CRESSIE

See Also

density.ppp, intensity for alternative estimates of point process intensity.

Examples

```
plot(nndensity(swedishpines))
```

---

nndist *Nearest neighbour distances*

Description

Computes the distance from each point to its nearest neighbour in a point pattern. Alternatively computes the distance to the second nearest neighbour, or third nearest, etc.

Usage

```
  nndist(X, ...)
  ## S3 method for class 'ppp'
nndist(X, ..., k=1, by=NULL, method="C")
  ## Default S3 method:
nndist(X, Y=NULL, ..., k=1, by=NULL, method="C")
```

Arguments

X,Y	Arguments specifying the locations of a set of points. For nndist.ppp, the argument X should be a point pattern (object of class "ppp"). For nndist.default, typically X and Y would be numeric vectors of equal length. Alternatively Y may be omitted and X may be a list with two components x and y, or a matrix with two columns.
...	Ignored by nndist.ppp and nndist.default.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.
by	Optional. A factor, which separates X into groups. The algorithm will compute the distance to the nearest point in each group.
method	String specifying which method of calculation to use. Values are "C" and "interpreted".

Details

This function computes the Euclidean distance from each point in a point pattern to its nearest neighbour (the nearest other point of the pattern). If `k` is specified, it computes the distance to the `k`th nearest neighbour.

The function `nndist` is generic, with a method for point patterns (objects of class `"ppp"`), and a default method for coordinate vectors. There is also a method for line segment patterns, `nndist.psp`.

The method for point patterns expects a single point pattern argument `X` and returns the vector of its nearest neighbour distances.

The default method expects that `X` and `Y` will determine the coordinates of a set of points. Typically `X` and `Y` would be numeric vectors of equal length. Alternatively `Y` may be omitted and `X` may be a list with two components named `x` and `y`, or a matrix or data frame with two columns.

The argument `k` may be a single integer, or an integer vector. If it is a vector, then the kth nearest neighbour distances are computed for each value of k specified in the vector.

If the argument `by` is given, it should be a `factor`, of length equal to the number of points in `X`. This factor effectively partitions `X` into subsets, each subset associated with one of the levels of `X`. The algorithm will then compute, for each point of `X`, the distance to the nearest neighbour *in each subset*.

The argument `method` is not normally used. It is retained only for checking the validity of the software. If `method = "interpreted"` then the distances are computed using interpreted R code only. If `method="C"` (the default) then C code is used. The C code is faster by two to three orders of magnitude and uses much less memory.

If there is only one point (if `x` has length 1), then a nearest neighbour distance of `Inf` is returned. If there are no points (if `x` has length zero) a numeric vector of length zero is returned.

To identify *which* point is the nearest neighbour of a given point, use `nnwhich`.

To use the nearest neighbour distances for statistical inference, it is often advisable to use the edge-corrected empirical distribution, computed by `Gest`.

To find the nearest neighbour distances from one point pattern to another point pattern, use `nncross`.

Value

Numeric vector or matrix containing the nearest neighbour distances for each point.

If `k = 1` (the default), the return value is a numeric vector `v` such that `v[i]` is the nearest neighbour distance for the `i`th data point.

If `k` is a single integer, then the return value is a numeric vector `v` such that `v[i]` is the `k`th nearest neighbour distance for the `i`th data point.

If `k` is a vector, then the return value is a matrix `m` such that `m[i,j]` is the `k[j]`th nearest neighbour distance for the `i`th data point.

If the argument `by` is given, then the result is a data frame containing the distances described above, from each point of `X`, to the nearest point in each subset of `X` defined by the factor `by`.

Nearest neighbours of each type

If `X` is a multitype point pattern and `by=marks(X)`, then the algorithm will compute, for each point of `X`, the distance to the nearest neighbour of each type. See the Examples.

To find the minimum distance from *any* point of type `i` to the nearest point of type `j`, for all combinations of `i` and `j`, use the R function `aggregate` as suggested in the Examples.

Warnings

An infinite or NA value is returned if the distance is not defined (e.g. if there is only one point in the point pattern).

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

nndist.psp, pairdist, Gest, nnwhich, nncross.

Examples

```
data(cells)
# nearest neighbours
d <- nndist(cells)

# second nearest neighbours
d2 <- nndist(cells, k=2)

# first, second and third nearest
d1to3 <- nndist(cells, k=1:3)

x <- runif(100)
y <- runif(100)
d <- nndist(x, y)

# Stienen diagram
plot(cells %mark% (nndist(cells)/2), markscale=1)

# distance to nearest neighbour of each type
nnda <- nndist(ants, by=marks(ants))
head(nnda)
# For nest number 1, the nearest Cataglyphis nest is 87.32125 units away

# Use of 'aggregate':
# minimum distance between each pair of types
aggregate(nnda, by=list(from=marks(ants)), min)
# Always a symmetric matrix

# mean nearest neighbour distances
aggregate(nnda, by=list(from=marks(ants)), mean)
# The mean distance from a Messor nest to
# the nearest Cataglyphis nest is 59.02549 units
```

---

nndist.lpp *Nearest neighbour distances on a linear network*

Description

Given a pattern of points on a linear network, compute the nearest-neighbour distances, measured by the shortest path in the network.

Usage

```
## S3 method for class 'lpp'
nndist(X, ..., k=1, method="C")
```

Arguments

X	Point pattern on linear network (object of class "lpp").
method	Optional string determining the method of calculation. Either "interpreted" or "C".
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.
...	Ignored.

Details

Given a pattern of points on a linear network, this function computes the nearest neighbour distance for each point (i.e. the distance from each point to the nearest other point), measuring distance by the shortest path in the network.

If method="C" the distances are computed using code in the C language. If method="interpreted" then the computation is performed using interpreted R code. The R code is much slower, but is provided for checking purposes.

The kth nearest neighbour distance is infinite if the kth nearest neighbour does not exist. This can occur if there are fewer than k+1 points in the dataset, or if the linear network is not connected.

Value

A numeric vector, of length equal to the number of points in X, or a matrix, with one row for each point in X and one column for each entry of k. Entries are nonnegative numbers or infinity (Inf).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

lpp

Examples

```
X <- runiflpp(12, simplenet)
nndist(X)
nndist(X, k=2)
```

nndist.pp3 *Nearest neighbour distances in three dimensions*

Description

Computes the distance from each point to its nearest neighbour in a three-dimensional point pattern. Alternatively computes the distance to the second nearest neighbour, or third nearest, etc.

Usage

```
## S3 method for class 'pp3'
nndist(X, ..., k=1)
```

Arguments

X	Three-dimensional point pattern (object of class "pp3").
...	Ignored.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.

Details

This function computes the Euclidean distance from each point in a three-dimensional point pattern to its nearest neighbour (the nearest other point of the pattern). If k is specified, it computes the distance to the kth nearest neighbour.

The function nndist is generic; this function nndist.pp3 is the method for the class "pp3".

The argument k may be a single integer, or an integer vector. If it is a vector, then the kth nearest neighbour distances are computed for each value of k specified in the vector.

If there is only one point (if x has length 1), then a nearest neighbour distance of Inf is returned. If there are no points (if x has length zero) a numeric vector of length zero is returned.

To identify *which* point is the nearest neighbour of a given point, use nnwhich.

To use the nearest neighbour distances for statistical inference, it is often advisable to use the edge-corrected empirical distribution, computed by G3est.

To find the nearest neighbour distances from one point pattern to another point pattern, use nncross.

Value

Numeric vector or matrix containing the nearest neighbour distances for each point.

If k = 1 (the default), the return value is a numeric vector v such that v[i] is the nearest neighbour distance for the ith data point.

If k is a single integer, then the return value is a numeric vector v such that v[i] is the kth nearest neighbour distance for the ith data point.

If k is a vector, then the return value is a matrix m such that m[i,j] is the k[j]th nearest neighbour distance for the ith data point.

Warnings

An infinite or NA value is returned if the distance is not defined (e.g. if there is only one point in the point pattern).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

based on code for two dimensions by Pavel Grabarnik

See Also

nndist, pairdist, G3est, nnwhich

Examples

```
X <- runifpoint3(40)

# nearest neighbours
d <- nndist(X)

# second nearest neighbours
d2 <- nndist(X, k=2)

# first, second and third nearest
d1to3 <- nndist(X, k=1:3)
```

---

nndist.ppx   *Nearest Neighbour Distances in Any Dimensions*

Description

Computes the distance from each point to its nearest neighbour in a multi-dimensional point pattern. Alternatively computes the distance to the second nearest neighbour, or third nearest, etc.

Usage

```
  ## S3 method for class 'ppx'
nndist(X, ..., k=1)
```

Arguments

X	Multi-dimensional point pattern (object of class "ppx").
...	Arguments passed to coords.ppx to determine which coordinates should be used.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.

Details

This function computes the Euclidean distance from each point in a multi-dimensional point pattern to its nearest neighbour (the nearest other point of the pattern). If k is specified, it computes the distance to the kth nearest neighbour.

The function nndist is generic; this function nndist.ppx is the method for the class "ppx".

The argument k may be a single integer, or an integer vector. If it is a vector, then the kth nearest neighbour distances are computed for each value of k specified in the vector.

If there is only one point (if x has length 1), then a nearest neighbour distance of Inf is returned. If there are no points (if x has length zero) a numeric vector of length zero is returned.

To identify *which* point is the nearest neighbour of a given point, use nnwhich.

To find the nearest neighbour distances from one point pattern to another point pattern, use nncross.

By default, both spatial and temporal coordinates are extracted. To obtain the spatial distance between points in a space-time point pattern, set temporal=FALSE.

Value

Numeric vector or matrix containing the nearest neighbour distances for each point.

If k = 1 (the default), the return value is a numeric vector v such that v[i] is the nearest neighbour distance for the ith data point.

If k is a single integer, then the return value is a numeric vector v such that v[i] is the kth nearest neighbour distance for the ith data point.

If k is a vector, then the return value is a matrix m such that m[i,j] is the k[j]th nearest neighbour distance for the ith data point.

Warnings

An infinite or NA value is returned if the distance is not defined (e.g. if there is only one point in the point pattern).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

nndist, pairdist, nnwhich

Examples

```
df <- data.frame(x=runif(5),y=runif(5),z=runif(5),w=runif(5))
X <- ppx(data=df)

# nearest neighbours
d <- nndist(X)

# second nearest neighbours
d2 <- nndist(X, k=2)

# first, second and third nearest
d1to3 <- nndist(X, k=1:3)
```

nndist.psp	*Nearest neighbour distances between line segments*

Description

Computes the distance from each line segment to its nearest neighbour in a line segment pattern. Alternatively finds the distance to the second nearest, third nearest etc.

Usage

```
  ## S3 method for class 'psp'
nndist(X, ..., k=1, method="C")
```

Arguments

X	A line segment pattern (object of class "psp").
...	Ignored.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.
method	String specifying which method of calculation to use. Values are "C" and "interpreted". Usually not specified.

Details

This is a method for the generic function nndist for the class "psp".

If k=1, this function computes the distance from each line segment to the nearest other line segment in X. In general it computes the distance from each line segment to the kth nearest other line segment. The argument k can also be a vector, and this computation will be performed for each value of k.

Distances are calculated using the Hausdorff metric. The Hausdorff distance between two line segments is the maximum distance from any point on one of the segments to the nearest point on the other segment.

If there are fewer than max(k)+1 line segments in the pattern, some of the nearest neighbour distances will be infinite (Inf).

The argument method is not normally used. It is retained only for checking the validity of the software. If method = "interpreted" then the distances are computed using interpreted R code only. If method="C" (the default) then compiled C code is used. The C code is somewhat faster.

Value

Numeric vector or matrix containing the nearest neighbour distances for each line segment.

If k = 1 (the default), the return value is a numeric vector v such that v[i] is the nearest neighbour distance for the ith segment.

If k is a single integer, then the return value is a numeric vector v such that v[i] is the kth nearest neighbour distance for the ith segment.

If k is a vector, then the return value is a matrix m such that m[i,j] is the k[j]th nearest neighbour distance for the ith segment.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

nndist, nndist.ppp

Examples

```
L <- psp(runif(10), runif(10), runif(10), runif(10), owin())
D <- nndist(L)
D <- nndist(L, k=1:3)
```

---

nnfromvertex *Nearest Data Point From Each Vertex in a Network*

Description

Given a point pattern on a linear network, for each vertex of the network find the nearest data point.

Usage

```
nnfromvertex(X, what = c("dist", "which"), k = 1)
```

Arguments

X	Point pattern on a linear network (object of class "lpp").
what	Character string specifying whether to return the nearest-neighbour distances, nearest-neighbour identifiers, or both.
k	Integer, or integer vector, specifying that the kth nearest neighbour should be returned.

Details

For each vertex (node) of the linear network, this algorithm finds the nearest data point to the vertex, and returns either the distance from the vertex to its nearest neighbour in X, or the serial number of the nearest neighbour in X, or both.

If k is an integer, then the k-th nearest neighbour is found instead.

If k is an integer vector, this is repeated for each integer in k.

Value

A numeric vector, matrix, or data frame.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

nndist.lpp

Examples

```
X <- runiflpp(5, simplenet)
nnfromvertex(X)
nnfromvertex(X, k=1:3)
```

---

nnfun *Nearest Neighbour Index Map as a Function*

Description

Compute the nearest neighbour index map of an object, and return it as a function.

Usage

```
nnfun(X, ...)

## S3 method for class 'ppp'
nnfun(X, ..., k=1)

## S3 method for class 'psp'
nnfun(X, ...)
```

Arguments

X	Any suitable dataset representing a two-dimensional collection of objects, such as a point pattern (object of class "ppp") or a line segment pattern (object of class "psp").
k	A single integer. The kth nearest neighbour will be found.
...	Extra arguments are ignored.

Details

For a collection X of two dimensional objects (such as a point pattern or a line segment pattern), the "nearest neighbour index function" of X is the mathematical function f such that, for any two-dimensional spatial location (x, y), the function value `f(x,y)` is the index i identifying the closest member of X. That is, if $i = f(x, y)$ then $X[i]$ is the closest member of the collection X to the location (x, y).

The command `f <- nnfun(X)` returns a *function* in the R language, with arguments `x,y`, that represents the nearest neighbour index function of `X`. Evaluating the function `f` in the form `v <- f(x,y)`, where `x` and `y` are any numeric vectors of equal length containing coordinates of spatial locations, yields the indices of the nearest neighbours to these locations.

If the argument `k` is specified then the `k`-th nearest neighbour will be found.

The result of `f <- nnfun(X)` also belongs to the class "funxy" and to the special class "nnfun". It can be printed and plotted immediately as shown in the Examples.

A `nnfun` object can be converted to a pixel image using `as.im`.

Value

A function with arguments x,y. The function also belongs to the class "nnfun" which has a method for print. It also belongs to the class "funxy" which has methods for plot, contour and persp.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

distfun, plot.funxy

Examples

```
f <- nnfun(cells)
f
plot(f)
f(0.2, 0.3)

g <- nnfun(cells, k=2)
g(0.2, 0.3)

L <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
h <- nnfun(L)
h(0.2, 0.3)
```

---

nnfun.lpp *Nearest Neighbour Map on Linear Network*

Description

Compute the nearest neighbour function of a point pattern on a linear network.

Usage

```
## S3 method for class 'lpp'
nnfun(X, ..., k=1)
```

Arguments

X	A point pattern on a linear network (object of class "lpp").
k	Integer. The algorithm finds the kth nearest neighbour in X from any spatial location.
...	Other arguments are ignored.

Details

The (geodesic) *nearest neighbour function* of a point pattern X on a linear network L tells us which point of X is closest to any given location.

If X is a point pattern on a linear network L, the *nearest neighbour function* of X is the mathematical function f defined for any location s on the network by f(s) = i, where X[i] is the closest point of X to the location s measured by the shortest path. In other words the value of f(s) is the identifier or serial number of the closest point of X.

The command nnfun.lpp is a method for the generic command nnfun for the class "lpp" of point patterns on a linear network.

If X is a point pattern on a linear network, f <- nnfun(X) returns a *function* in the R language, with arguments x,y, ..., that represents the nearest neighbour function of X. Evaluating the function f in the form v <- f(x,y), where x and y are any numeric vectors of equal length containing coordinates of spatial locations, yields a vector of identifiers or serial numbers of the data points closest to these spatial locations. More efficiently f can take the arguments x, y, seg, tp where seg and tp are the local coordinates on the network.

The result of f <- nnfun(X) also belongs to the class "linfun". It can be printed and plotted immediately as shown in the Examples. It can be converted to a pixel image using as.linim.

Value

A function in the R language, with arguments x,y and optional arguments seg,tp. It also belongs to the class "linfun" which has methods for plot, print etc.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

linfun, methods.linfun.

To compute the *distance* to the nearest neighbour, see distfun.lpp.

Examples

```
data(letterR)
X <- runiflpp(3, simplenet)
f <- nnfun(X)
f
plot(f)
```

---

nnmap *K-th Nearest Point Map*

Description

Given a point pattern, this function constructs pixel images giving the distance from each pixel to its k-th nearest neighbour in the point pattern, and the index of the k-th nearest neighbour.

Usage

```
nnmap(X, k = 1, what = c("dist", "which"),
...., W = as.owin(X),
is.sorted.X = FALSE, sortby = c("range", "var", "x", "y"))
```

Arguments

X	Point pattern (object of class "ppp").
k	Integer, or integer vector. The algorithm will find the kth nearest neighbour.
what	Character string specifying what information should be returned. Either the nearest neighbour distance ("dist"), the index of the nearest neighbour ("which"), or both.
...	Arguments passed to as.mask to determine the pixel resolution of the result.
W	Window (object of class "owin") specifying the spatial domain in which the distances will be computed. Defaults to the window of X.
is.sorted.X	Logical value attesting whether the point pattern X has been sorted. See Details.
sortby	Determines which coordinate to use to sort the point pattern. See Details.

Details

Given a point pattern X, this function constructs two pixel images:

- a distance map giving, for each pixel, the distance to the nearest point of X;
- a nearest neighbour map giving, for each pixel, the identifier of the nearest point of X.

If the argument k is specified, then the k-th nearest neighbours will be found.

If what="dist" then only the distance map is returned. If what="which" then only the nearest neighbour map is returned.

The argument k may be an integer or an integer vector. If it is a single integer, then the k-th nearest neighbours are computed. If it is a vector, then the k[i]-th nearest neighbours are computed for each entry k[i]. For example, setting k=1:3 will compute the nearest, second-nearest and third-nearest neighbours.

Value

A pixel image, or a list of pixel images.

By default (if what=c("dist", "which")), the result is a list with two components dist and which containing the distance map and the nearest neighbour map.

If what="dist" then the result is a real-valued pixel image containing the distance map.

If what="which" then the result is an integer-valued pixel image containing the nearest neighbour map.

If k is a vector of several integers, then the result is similar except that each pixel image is replaced by a list of pixel images, one for each entry of k.

Sorting data and pre-sorted data

Read this section if you care about the speed of computation.

For efficiency, the algorithm sorts the point pattern `X` into increasing order of the x coordinate or increasing order of the the y coordinate. Sorting is only an intermediate step; it does not affect the output, which is always given in the same order as the original data.

By default (if `sortby="range"`), the sorting will occur on the coordinate that has the larger range of values (according to the frame of the enclosing window of `X`). If `sortby = "var"`), sorting will occur on the coordinate that has the greater variance (in the pattern `X`). Setting `sortby="x"` or `sortby = "y"` will specify that sorting should occur on the x or y coordinate, respectively.

If the point pattern `X` is already sorted, then the argument `is.sorted.X` should be set to `TRUE`, and `sortby` should be set equal to `"x"` or `"y"` to indicate which coordinate is sorted.

Warning About Ties

Ties are possible: there may be two data points which lie exactly the same distance away from a particular pixel. This affects the results from `nnmap(what="which")`. The handling of ties is not well-defined: it is not consistent between different computers and different installations of R. If there are ties, then different calls to `nnmap(what="which")` may give inconsistent results. For example, you may get a different answer from `nnmap(what="which",k=1)` and `nnmap(what="which", k=1:2)[[1]]`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz> , and Jens Oehlschlaegel

See Also

`distmap`

Examples

```
plot(nnmap(cells, 2, what="which"))
```

---

nnmark *Mark of Nearest Neighbour*

Description

Given a marked point pattern dataset `X` this function computes, for each desired location `y`, the mark attached to the nearest neighbour of `y` in `X`. The desired locations `y` can be either a pixel grid or the point pattern `X` itself.

Usage

```
nnmark(X, ..., k = 1, at=c("pixels", "points"))
```

Arguments

X	A marked point pattern (object of class "ppp").
...	Arguments passed to as.mask to determine the pixel resolution.
k	Single integer. The kth nearest data point will be used.
at	String specifying whether to compute the values at a grid of pixel locations (at="pixels") or only at the points of X (at="points").

Details

Given a marked point pattern dataset X this function computes, for each desired location y, the mark attached to the point of X that is nearest to y. The desired locations y can be either a pixel grid or the point pattern X itself.

The argument X must be a marked point pattern (object of class "ppp", see ppp.object). The marks are allowed to be a vector or a data frame.

- If at="points", then for each point in X, the algorithm finds the nearest *other* point in X, and extracts the mark attached to it. The result is a vector or data frame containing the marks of the neighbours of each point.
- If at="pixels" (the default), then for each pixel in a rectangular grid, the algorithm finds the nearest point in X, and extracts the mark attached to it. The result is an image or a list of images containing the marks of the neighbours of each pixel. The pixel resolution is controlled by the arguments ... passed to as.mask.

If the argument k is given, then the k-th nearest neighbour will be used.

Value

If X *has a single column of marks:*

- If at="pixels" (the default), the result is a pixel image (object of class "im"). The value at each pixel is the mark attached to the nearest point of X.
- If at="points", the result is a vector or factor of length equal to the number of points in X. Entries are the mark values of the nearest neighbours of each point of X.

If X *has a data frame of marks:*

- If at="pixels" (the default), the result is a named list of pixel images (object of class "im"). There is one image for each column of marks. This list also belongs to the class "solist", for which there is a plot method.
- If at="points", the result is a data frame with one row for each point of X, Entries are the mark values of the nearest neighbours of each point of X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

Smooth.ppp, marktable, nnwhich

Examples

```
plot(nnmark(ants))
v <- nnmark(ants, at="points")
v[1:10]
plot(nnmark(finpines))
vf <- nnmark(finpines, at="points")
vf[1:5,]
```

---

nnorient *Nearest Neighbour Orientation Distribution*

Description

Computes the distribution of the orientation of the vectors from each point to its nearest neighbour.

Usage

```
nnorient(X, ..., cumulative = FALSE, correction, k = 1,
                 unit = c("degree", "radian"),
                 domain = NULL, ratio = FALSE)
```

Arguments

X	Point pattern (object of class "ppp").
...	Arguments passed to circdensity to control the kernel smoothing, if cumulative=FALSE.
cumulative	Logical value specifying whether to estimate the probability density (cumulative=FALSE, the default) or the cumulative distribution function (cumulative=TRUE).
correction	Character vector specifying edge correction or corrections. Options are "none", "bord.modif", "good" and "best". Alternatively correction="all" selects all options.
k	Integer. The kth nearest neighbour will be used.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
unit	Unit in which the angles should be expressed. Either "degree" or "radian".
domain	Optional window. The first point x_i of each pair of points will be constrained to lie in domain.

Details

This algorithm considers each point in the pattern X and finds its nearest neighbour (or kth nearest neighour). The *direction* of the arrow joining the data point to its neighbour is measured, as an angle in degrees or radians, anticlockwise from the x axis.

If cumulative=FALSE (the default), a kernel estimate of the probability density of the angles is calculated using circdensity. This is the function $\vartheta(\phi)$ defined in Illian et al (2008), equation (4.5.3), page 253.

If cumulative=TRUE, then the cumulative distribution function of these angles is calculated.

In either case the result can be plotted as a rose diagram by rose, or as a function plot by plot.fv.

The algorithm gives each observed direction a weight, determined by an edge correction, to adjust for the fact that some interpoint distances are more likely to be observed than others. The choice of edge correction or corrections is determined by the argument `correction`.

It is also possible to calculate an estimate of the probability density from the cumulative distribution function, by numerical differentiation. Use `deriv.fv` with the argument `Dperiodic=TRUE`.

Value

A function value table (object of class `"fv"`) containing the estimates of the probability density or the cumulative distribution function of angles, in degrees (if `unit="degree"`) or radians (if `unit="radian"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Illian, J., Penttinen, A., Stoyan, H. and Stoyan, D. (2008) *Statistical Analysis and Modelling of Spatial Point Patterns.* Wiley.

See Also

`pairorient`

Examples

```
rose(nnorient(redwood, adjust=0.6), col="grey")
plot(CDF <- nnorient(redwood, cumulative=TRUE))
```

`nnwhich` *Nearest neighbour*

Description

Finds the nearest neighbour of each point in a point pattern.

Usage

```
  nnwhich(X, ...)
  ## S3 method for class 'ppp'
nnwhich(X, ..., k=1, by=NULL, method="C")
  ## Default S3 method:
nnwhich(X, Y=NULL, ..., k=1, by=NULL, method="C")
```

Arguments

X,Y	Arguments specifying the locations of a set of points. For nnwhich.ppp, the argument X should be a point pattern (object of class "ppp"). For nnwhich.default, typically X and Y would be numeric vectors of equal length. Alternatively Y may be omitted and X may be a list with two components x and y, or a matrix with two columns.
...	Ignored by nnwhich.ppp and nnwhich.default.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.
by	Optional. A factor, which separates X into groups. The algorithm will find the nearest neighbour in each group.
method	String specifying which method of calculation to use. Values are "C" and "interpreted".

Details

For each point in the given point pattern, this function finds its nearest neighbour (the nearest other point of the pattern). By default it returns a vector giving, for each point, the index of the point's nearest neighbour. If k is specified, the algorithm finds each point's kth nearest neighbour.

The function nnwhich is generic, with method for point patterns (objects of class "ppp") and a default method which are described here, as well as a method for three-dimensional point patterns (objects of class "pp3", described in nnwhich.pp3.

The method nnwhich.ppp expects a single point pattern argument X. The default method expects that X and Y will determine the coordinates of a set of points. Typically X and Y would be numeric vectors of equal length. Alternatively Y may be omitted and X may be a list with two components named x and y, or a matrix or data frame with two columns.

The argument k may be a single integer, or an integer vector. If it is a vector, then the kth nearest neighbour distances are computed for each value of k specified in the vector.

If the argument by is given, it should be a factor, of length equal to the number of points in X. This factor effectively partitions X into subsets, each subset associated with one of the levels of X. The algorithm will then find, for each point of X, the nearest neighbour *in each subset*.

If there are no points (if x has length zero) a numeric vector of length zero is returned. If there is only one point (if x has length 1), then the nearest neighbour is undefined, and a value of NA is returned. In general if the number of points is less than or equal to k, then a vector of NA's is returned.

The argument method is not normally used. It is retained only for checking the validity of the software. If method = "interpreted" then the distances are computed using interpreted R code only. If method="C" (the default) then C code is used. The C code is faster by two to three orders of magnitude and uses much less memory.

To evaluate the *distance* between a point and its nearest neighbour, use nndist.

To find the nearest neighbours from one point pattern to another point pattern, use nncross.

Value

Numeric vector or matrix giving, for each point, the index of its nearest neighbour (or kth nearest neighbour).

If k = 1 (the default), the return value is a numeric vector v giving the indices of the nearest neighbours (the nearest neighbout of the ith point is the jth point where j = v[i]).

If k is a single integer, then the return value is a numeric vector giving the indices of the kth nearest neighbours.

If k is a vector, then the return value is a matrix m such that m[i,j] is the index of the k[j]th nearest neighbour for the ith data point.

If the argument by is given, then the result is a data frame containing the indices described above, from each point of X, to the nearest point in each subset of X defined by the factor by.

Nearest neighbours of each type

If X is a multitype point pattern and by=marks(X), then the algorithm will find, for each point of X, the nearest neighbour of each type. See the Examples.

Warnings

A value of NA is returned if there is only one point in the point pattern.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

nndist, nncross

Examples

```
data(cells)
plot(cells)
m <- nnwhich(cells)
m2 <- nnwhich(cells, k=2)

# plot nearest neighbour links
b <- cells[m]
arrows(cells$x, cells$y, b$x, b$y, angle=15, length=0.15, col="red")

# find points which are the neighbour of their neighbour
self <- (m[m] == seq(m))
# plot them
A <- cells[self]
B <- cells[m[self]]
plot(cells)
segments(A$x, A$y, B$x, B$y)

# nearest neighbours of each type
head(nnwhich(ants, by=marks(ants)))
```

nnwhich.lpp *Identify Nearest Neighbours on a Linear Network*

Description

Given a pattern of points on a linear network, identify the nearest neighbour for each point, measured by the shortest path in the network.

Usage

```
## S3 method for class 'lpp'
nnwhich(X, ..., k=1, method="C")
```

Arguments

X	Point pattern on linear network (object of class "lpp").
method	Optional string determining the method of calculation. Either "interpreted" or "C".
k	Integer, or integer vector. The algorithm will find the kth nearest neighbour.
...	Ignored.

Details

Given a pattern of points on a linear network, this function finds the nearest neighbour of each point (i.e. for each point it identifies the nearest other point) measuring distance by the shortest path in the network.

If method="C" the task is performed using code in the C language. If method="interpreted" then the computation is performed using interpreted R code. The R code is much slower, but is provided for checking purposes.

The result is NA if the kth nearest neighbour does not exist. This can occur if there are fewer than k+1 points in the dataset, or if the linear network is not connected.

Value

An integer vector, of length equal to the number of points in X, identifying the nearest neighbour of each point. If nnwhich(X)[2] = 4 then the nearest neighbour of point 2 is point 4.

Alternatively a matrix with one row for each point in X and one column for each entry of k.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

lpp

Examples

```
X <- runiflpp(10, simplenet)
nnwhich(X)
nnwhich(X, k=2)
```

`nnwhich.pp3` *Nearest neighbours in three dimensions*

Description

Finds the nearest neighbour of each point in a three-dimensional point pattern.

Usage

```
## S3 method for class 'pp3'
nnwhich(X, ..., k=1)
```

Arguments

`X`	Three-dimensional point pattern (object of class `"pp3"`).
`...`	Ignored.
`k`	Integer, or integer vector. The algorithm will compute the distance to the `k`th nearest neighbour.

Details

For each point in the given three-dimensional point pattern, this function finds its nearest neighbour (the nearest other point of the pattern). By default it returns a vector giving, for each point, the index of the point's nearest neighbour. If `k` is specified, the algorithm finds each point's `k`th nearest neighbour.

The function `nnwhich` is generic. This is the method for the class `"pp3"`.

If there are no points in the pattern, a numeric vector of length zero is returned. If there is only one point, then the nearest neighbour is undefined, and a value of `NA` is returned. In general if the number of points is less than or equal to `k`, then a vector of `NA`'s is returned.

To evaluate the *distance* between a point and its nearest neighbour, use `nndist`.

To find the nearest neighbours from one point pattern to another point pattern, use `nncross`.

Value

Numeric vector or matrix giving, for each point, the index of its nearest neighbour (or `k`th nearest neighbour).

If `k = 1` (the default), the return value is a numeric vector `v` giving the indices of the nearest neighbours (the nearest neighbout of the `i`th point is the `j`th point where `j = v[i]`).

If `k` is a single integer, then the return value is a numeric vector giving the indices of the `k`th nearest neighbours.

If `k` is a vector, then the return value is a matrix `m` such that `m[i,j]` is the index of the `k[j]`th nearest neighbour for the `i`th data point.

Warnings

A value of `NA` is returned if there is only one point in the point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

based on two-dimensional code by Pavel Grabarnik

See Also

nnwhich, nndist, nncross

Examples

```
X <- runifpoint3(30)
m <- nnwhich(X)
m2 <- nnwhich(X, k=2)
```

---

nnwhich.ppx *Nearest Neighbours in Any Dimensions*

Description

Finds the nearest neighbour of each point in a multi-dimensional point pattern.

Usage

```
  ## S3 method for class 'ppx'
nnwhich(X, ..., k=1)
```

Arguments

X	Multi-dimensional point pattern (object of class "ppx").
...	Arguments passed to coords.ppx to determine which coordinates should be used.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.

Details

For each point in the given multi-dimensional point pattern, this function finds its nearest neighbour (the nearest other point of the pattern). By default it returns a vector giving, for each point, the index of the point's nearest neighbour. If k is specified, the algorithm finds each point's kth nearest neighbour.

The function nnwhich is generic. This is the method for the class "ppx".

If there are no points in the pattern, a numeric vector of length zero is returned. If there is only one point, then the nearest neighbour is undefined, and a value of NA is returned. In general if the number of points is less than or equal to k, then a vector of NA's is returned.

To evaluate the *distance* between a point and its nearest neighbour, use nndist.

To find the nearest neighbours from one point pattern to another point pattern, use nncross.

By default, both spatial and temporal coordinates are extracted. To obtain the spatial distance between points in a space-time point pattern, set temporal=FALSE.

Value

Numeric vector or matrix giving, for each point, the index of its nearest neighbour (or kth nearest neighbour).

If k = 1 (the default), the return value is a numeric vector v giving the indices of the nearest neighbours (the nearest neighbout of the ith point is the jth point where j = v[i]).

If k is a single integer, then the return value is a numeric vector giving the indices of the kth nearest neighbours.

If k is a vector, then the return value is a matrix m such that m[i,j] is the index of the k[j]th nearest neighbour for the ith data point.

Warnings

A value of NA is returned if there is only one point in the point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

nnwhich, nndist, nncross

Examples

```
df <- data.frame(x=runif(5),y=runif(5),z=runif(5),w=runif(5))
X <- ppx(data=df)
m <- nnwhich(X)
m2 <- nnwhich(X, k=2)
```

---

nobjects *Count Number of Geometrical Objects in a Spatial Dataset*

Description

A generic function to count the number of geometrical objects in a spatial dataset.

Usage

```
nobjects(x)

## S3 method for class 'ppp'
nobjects(x)

## S3 method for class 'ppx'
nobjects(x)

## S3 method for class 'psp'
nobjects(x)

## S3 method for class 'tess'
nobjects(x)
```

Arguments

x   A dataset.

Details

The generic function `nobjects` counts the number of geometrical objects in the spatial dataset `x`.

The methods for point patterns (classes `"ppp"` and `"ppx"`, embracing `"pp3"` and `"lpp"`) count the number of points in the pattern.

The method for line segment patterns (class `"psp"`) counts the number of line segments in the pattern.

The method for tessellations (class `"tess"`) counts the number of tiles of the tessellation.

Value

A single integer.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`npoints`

Examples

```
nobjects(redwood)
nobjects(edges(letterR))
nobjects(dirichlet(cells))
```

---

npfun   *Dummy Function Returns Number of Points*

Description

Returns a summary function which is constant with value equal to the number of points in the point pattern.

Usage

```
npfun(X, ..., r)
```

Arguments

`X`	Point pattern.
`...`	Ignored.
`r`	Vector of values of the distance argument r.

Details

This function is normally not called by the user. Instead it is passed as an argument to the function `psst`.

Value

Object of class "fv" representing a constant function.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Ege Rubak <rubak@math.aau.dk> and Jesper Møller.

See Also

`psst`

Examples

```
fit0 <- ppm(cells, ~1, nd=10)
v <- psst(fit0, npfun)
```

`npoints` *Number of Points in a Point Pattern*

Description

Returns the number of points in a point pattern of any kind.

Usage

```
  npoints(x)
  ## S3 method for class 'ppp'
npoints(x)
  ## S3 method for class 'pp3'
npoints(x)
  ## S3 method for class 'ppx'
npoints(x)
```

Arguments

x	A point pattern (object of class "ppp", "pp3", "ppx" or some other suitable class).

Details

This function returns the number of points in a point pattern. The function `npoints` is generic with methods for the classes "ppp", "pp3", "ppx" and possibly other classes.

Value

Integer.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppp.object, print.pp3, print.ppx.

Examples

```
data(cells)
npoints(cells)
```

---

nsegments — *Number of Line Segments in a Line Segment Pattern*

Description

Returns the number of line segments in a line segment pattern.

Usage

```
nsegments(x)

## S3 method for class 'psp'
nsegments(x)
```

Arguments

x	A line segment pattern, i.e. an object of class psp, or an object containing a linear network.

Details

This function is generic, with methods for classes psp, linnet and lpp.

Value

Integer.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

npoints(), psp.object()

Examples

```
nsegments(copper$Lines)
nsegments(copper$SouthLines)
```

nvertices *Count Number of Vertices*

Description

Count the number of vertices in an object for which vertices are well-defined.

Usage

```
nvertices(x, ...)

## S3 method for class 'owin'
nvertices(x, ...)

## Default S3 method:
nvertices(x, ...)
```

Arguments

x	A window (object of class "owin"), or some other object which has vertices.
...	Currently ignored.

Details

This function counts the number of vertices of x as they would be returned by vertices(x). It is more efficient than executing npoints(vertices(x)).

Value

A single integer.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> and Suman Rakshit.

See Also

vertices

Examples

```
nvertices(square(2))
nvertices(letterR)
```

objsurf *Objective Function Surface*

Description

For a model that was fitted by optimisation, compute the values of the objective function in a neighbourhood of the optimal value.

Usage

```
objsurf(x, ...)

## S3 method for class 'dppm'
objsurf(x, ..., ngrid = 32, ratio = 1.5, verbose = TRUE)

## S3 method for class 'kppm'
objsurf(x, ..., ngrid = 32, ratio = 1.5, verbose = TRUE)

## S3 method for class 'minconfit'
objsurf(x, ..., ngrid = 32, ratio = 1.5, verbose = TRUE)
```

Arguments

x	Some kind of model that was fitted by finding the optimal value of an objective function. An object of class "dppm", "kppm" or "minconfit".
...	Extra arguments are usually ignored.
ngrid	Number of grid points to evaluate along each axis. Either a single integer, or a pair of integers. For example ngrid=32 would mean a 32 * 32 grid.
ratio	Number greater than 1 determining the range of parameter values to be considered. If the optimal parameter value is opt then the objective function will be evaluated for values between opt/ratio and opt * ratio.
verbose	Logical value indicating whether to print progress reports.

Details

The object x should be some kind of model that was fitted by maximising or minimising the value of an objective function. The objective function will be evaluated on a grid of values of the model parameters.

Currently the following types of objects are accepted:

- an object of class "dppm" representing a determinantal point process. See dppm.
- an object of class "kppm" representing a cluster point process or Cox point process. See kppm.
- an object of class "minconfit" representing a minimum-contrast fit between a summary function and its theoretical counterpart. See mincontrast.

The result is an object of class "objsurf" which can be printed and plotted: see methods.objsurf.

Value

An object of class "objsurf" which can be printed and plotted. Essentially a list containing entries x, y, z giving the parameter values and objective function values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Ege Rubak <rubak@math.aau.dk>.

See Also

methods.objsurf, kppm, mincontrast

Examples

```
fit <- kppm(redwood ~ 1, "Thomas")
os <- objsurf(fit)

if(interactive()) {
  plot(os)
  contour(os, add=TRUE)
  persp(os)
}
```

---

opening *Morphological Opening*

Description

Perform morphological opening of a window, a line segment pattern or a point pattern.

Usage

```
 opening(w, r, ...)

 ## S3 method for class 'owin'
opening(w, r, ..., polygonal=NULL)

 ## S3 method for class 'ppp'
opening(w, r, ...)

 ## S3 method for class 'psp'
opening(w, r, ...)
```

Arguments

w	A window (object of class "owin" or a line segment pattern (object of class "psp") or a point pattern (object of class "ppp").
r	positive number: the radius of the opening.
...	extra arguments passed to as.mask controlling the pixel resolution, if a pixel approximation is used
polygonal	Logical flag indicating whether to compute a polygonal approximation to the erosion (polygonal=TRUE) or a pixel grid approximation (polygonal=FALSE).

Details

The morphological opening (Serra, 1982) of a set W by a distance $r > 0$ is the subset of points in W that can be separated from the boundary of W by a circle of radius r. That is, a point x belongs to the opening if it is possible to draw a circle of radius r (not necessarily centred on x) that has x on the inside and the boundary of W on the outside. The opened set is a subset of `W`.

For a small radius r, the opening operation has the effect of smoothing out irregularities in the boundary of W. For larger radii, the opening operation removes promontories in the boundary. For very large radii, the opened set is empty.

The algorithm applies `erosion` followed by `dilation`.

Value

If `r > 0`, an object of class `"owin"` representing the opened region. If `r=0`, the result is identical to `W`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Serra, J. (1982) Image analysis and mathematical morphology. Academic Press.

See Also

`closing` for the opposite operation.

`dilation`, `erosion` for the basic operations.

`owin`, `as.owin` for information about windows.

Examples

```
v <- opening(letterR, 0.3)
plot(letterR, type="n", main="opening")
plot(v, add=TRUE, col="grey")
plot(letterR, add=TRUE)
```

`Ops.msr` *Arithmetic Operations on Measures*

Description

These group generic methods for the class `"msr"` allow the arithmetic operators `+`, `-`, `*` and `/` to be applied directly to measures.

Usage

```
## S3 methods for group generics have prototypes:
Ops(e1, e2)
```

Arguments

`e1, e2` objects of class "msr".

Details

Arithmetic operators on a measure `A` are only defined in some cases. The arithmetic operator is effectively applied to the value of `A(W)` for every spatial domain `W`. If the result is a measure, then this operation is valid.

If `A` is a measure (object of class "msr") then the operations `-A` and `+A` are defined.

If `A` and `B` are measures with the same dimension (i.e. both are scalar-valued, or both are k-dimensional vector-valued) then `A + B` and `A - B` are defined.

If `A` is a measure and `z` is a numeric value, then `A * z` and `A / z` are defined, and `z * A` is defined.

Value

Another measure (object of class "msr").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

with.msr

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
rp <- residuals(fit, type="pearson")
rp

-rp
2 * rp
rp /2

rp - rp

rr <- residuals(fit, type="raw")
rp - rr
```

`Ord` *Generic Ord Interaction model*

Description

Creates an instance of an Ord-type interaction point process model which can then be fitted to point pattern data.

Usage

```
Ord(pot, name)
```

Arguments

`pot`	An S language function giving the user-supplied interaction potential.
`name`	Character string.

Details

Ord's point process model (Ord, 1977) is a Gibbs point process of infinite order. Each point x_i in the point pattern x contributes a factor $g(a_i)$ where $a_i = a(x_i, x)$ is the area of the tile associated with x_i in the Dirichlet tessellation of x.

Ord (1977) proposed fitting this model to forestry data when $g(a)$ has a simple "threshold" form. That model is implemented in our function `OrdThresh`. The present function `Ord` implements the case of a completely general Ord potential $g(a)$ specified as an S language function `pot`.

This is experimental.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Ord, J.K. (1977) Contribution to the discussion of Ripley (1977).

Ord, J.K. (1978) How many trees in a forest? *Mathematical Scientist* **3**, 23–33.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

`ppm`, `ppm.object`, `OrdThresh`

ord.family	*Ord Interaction Process Family*

Description

An object describing the family of all Ord interaction point processes

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the family of point process models introduced by Ord (1977).

If you need to create a specific Ord-type model for use in analysis, use the function `OrdThresh` or `Ord`.

Anyway, `ord.family` is an object of class `"isf"` containing a function `ord.family$eval` for evaluating the sufficient statistics of any Ord type point process model taking an exponential family form.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Ord, J.K. (1977) Contribution to the discussion of Ripley (1977).

Ord, J.K. (1978) How many trees in a forest? *Mathematical Scientist* **3**, 23–33.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

`pairwise.family`, `pairsat.family`, `Poisson`, `Pairwise`, `PairPiece`, `Strauss`, `StraussHard`, `Softcore`, `Geyer`, `SatPiece`, `Saturated`, `Ord`, `OrdThresh`

OrdThresh	*Ord's Interaction model*

Description

Creates an instance of Ord's point process model which can then be fitted to point pattern data.

Usage

```
OrdThresh(r)
```

Arguments

`r` Positive number giving the threshold value for Ord's model.

Details

Ord's point process model (Ord, 1977) is a Gibbs point process of infinite order. Each point x_i in the point pattern x contributes a factor $g(a_i)$ where $a_i = a(x_i, x)$ is the area of the tile associated with x_i in the Dirichlet tessellation of x. The function g is simply $g(a) = 1$ if $a \geq r$ and $g(a) = \gamma < 1$ if $a < r$, where r is called the threshold value.

This function creates an instance of Ord's model with a given value of r. It can then be fitted to point process data using `ppm`.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Ord, J.K. (1977) Contribution to the discussion of Ripley (1977).

Ord, J.K. (1978) How many trees in a forest? *Mathematical Scientist* **3**, 23–33.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

`ppm`, `ppm.object`

`overlap.owin` *Compute Area of Overlap*

Description

Computes the area of the overlap (intersection) of two windows.

Usage

```
overlap.owin(A, B)
```

Arguments

`A,B` Windows (objects of class `"owin"`).

Details

This function computes the area of the overlap between the two windows A and B.

If one of the windows is a binary mask, then both windows are converted to masks on the same grid, and the area is computed by counting pixels. Otherwise, the area is computed analytically (using the discrete Stokes theorem).

Value

A single numeric value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

intersect.owin, area.owin, setcov.

Examples

```
A <- square(1)
B <- shift(A, c(0.3, 0.2))
overlap.owin(A, B)
```

---

owin *Create a Window*

Description

Creates an object of class "owin" representing an observation window in the two-dimensional plane

Usage

```
  owin(xrange=c(0,1), yrange=c(0,1), ..., poly=NULL, mask=NULL,
unitname=NULL, xy=NULL)
```

Arguments

xrange	x coordinate limits of enclosing box
yrange	y coordinate limits of enclosing box
...	Ignored.
poly	Optional. Polygonal boundary of window. Incompatible with mask.
mask	Optional. Logical matrix giving binary image of window. Incompatible with poly.
unitname	Optional. Name of unit of length. Either a single character string, or a vector of two character strings giving the singular and plural forms, respectively.
xy	Optional. List with components x and y specifying the pixel coordinates for mask.

Details

In the **spatstat** library, a point pattern dataset must include information about the window of observation. This is represented by an object of class "owin". See owin.object for an overview.

To create a window in its own right, users would normally invoke owin, although sometimes as.owin may be convenient.

A window may be rectangular, polygonal, or a mask (a binary image).

- **rectangular windows:** If only xrange and yrange are given, then the window will be rectangular, with its x and y coordinate dimensions given by these two arguments (which must be vectors of length 2). If no arguments are given at all, the default is the unit square with dimensions xrange=c(0,1) and yrange=c(0,1).
- **polygonal windows:** If poly is given, then the window will be polygonal.
  - *single polygon:* If poly is a matrix or data frame with two columns, or a structure with two component vectors x and y of equal length, then these values are interpreted as the cartesian coordinates of the vertices of a polygon circumscribing the window. The vertices must be listed *anticlockwise*. No vertex should be repeated (i.e. do not repeat the first vertex).
  - *multiple polygons or holes:* If poly is a list, each entry poly[[i]] of which is a matrix or data frame with two columns or a structure with two component vectors x and y of equal length, then the successive list members poly[[i]] are interpreted as separate polygons which together make up the boundary of the window. The vertices of each polygon must be listed *anticlockwise* if the polygon is part of the external boundary, but *clockwise* if the polygon is the boundary of a hole in the window. Again, do not repeat any vertex.
- **binary masks:** If mask is given, then the window will be a binary image.
  - *Specified by logical matrix:* Normally the argument mask should be a logical matrix such that mask[i,j] is TRUE if the point (x[j],y[i]) belongs to the window, and FALSE if it does not. Note carefully that rows of mask correspond to the y coordinate, and columns to the x coordinate. Here x and y are vectors of x and y coordinates equally spaced over xrange and yrange respectively. The pixel coordinate vectors x and y may be specified explicitly using the argument xy, which should be a list containing components x and y. Alternatively there is a sensible default.
  - *Specified by list of pixel coordinates:* Alternatively the argument mask can be a data frame with 2 or 3 columns. If it has 2 columns, it is expected to contain the spatial coordinates of all the pixels which are inside the window. If it has 3 columns, it should contain the spatial coordinates (x, y) of every pixel in the grid, and the logical value associated with each pixel. The pixels may be listed in any order.

To create a window which is mathematically defined by inequalities in the Cartesian coordinates, use raster.x() and raster.y() as in the examples below.

Functions square and disc will create square and circular windows, respectively.

Value

An object of class "owin" describing a window in the two-dimensional plane.

Validity of polygon data

Polygon data may contain geometrical inconsistencies such as self-intersections and overlaps. These inconsistencies must be removed to prevent problems in other **spatstat** functions. By default, polygon data will be repaired automatically using polygon-clipping code. The repair process may change the number of vertices in a polygon and the number of polygon components. To disable the repair process, set spatstat.options(fixpolygons=FALSE).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin.object, as.owin, complement.owin, ppp.object, ppp
square, hexagon, regularpolygon, disc, ellipse.

Examples

```
  w <- owin()
  w <- owin(c(0,1), c(0,1))
  # the unit square

  w <- owin(c(10,20), c(10,30), unitname=c("foot","feet"))
  # a rectangle of dimensions 10 x 20 feet
  # with lower left corner at (10,10)

  # polygon (diamond shape)
  w <- owin(poly=list(x=c(0.5,1,0.5,0),y=c(0,1,2,1)))
  w <- owin(c(0,1), c(0,2), poly=list(x=c(0.5,1,0.5,0),y=c(0,1,2,1)))

  # polygon with hole
  ho <- owin(poly=list(list(x=c(0,1,1,0), y=c(0,0,1,1)),
                       list(x=c(0.6,0.4,0.4,0.6), y=c(0.2,0.2,0.4,0.4))))

  w <- owin(c(-1,1), c(-1,1), mask=matrix(TRUE, 100,100))
          # 100 x 100 image, all TRUE
  X <- raster.x(w)
  Y <- raster.y(w)
  wm <- owin(w$xrange, w$yrange, mask=(X^2 + Y^2 <= 1))
          # discrete approximation to the unit disc

  ## Not run:
  if(FALSE) {
    plot(c(0,1),c(0,1),type="n")
    bdry <- locator()
    # click the vertices of a polygon (anticlockwise)
  }

## End(Not run)

  w <- owin(poly=bdry)
  ## Not run: plot(w)

 ## Not run:
 im <- as.logical(matrix(scan("myfile"), nrow=128, ncol=128))
 # read in an arbitrary 128 x 128 digital image from text file
 rim <- im[, 128:1]
 # Assuming it was given in row-major order in the file
 # i.e. scanning left-to-right in rows from top-to-bottom,
 # the use of matrix() has effectively transposed rows & columns,
 # so to convert it to our format just reverse the column order.
 w <- owin(mask=rim)
```

```
  plot(w)
  # display it to check!

## End(Not run)
```

owin.object *Class owin*

Description

A class owin to define the "observation window" of a point pattern

Details

In the **spatstat** library, a point pattern dataset must include information about the window or region in which the pattern was observed. A window is described by an object of class "owin". Windows of arbitrary shape are supported.

An object of class "owin" has one of three types:

"rectangle":	a rectangle in the two-dimensional plane with edges parallel to the axes
"polygonal":	a region whose boundary is a polygon or several polygons. The region may have holes and may consist c
"mask":	a binary image (a logical matrix) set to TRUE for pixels inside the window and FALSE outside the window

Objects of class "owin" may be created by the function owin and converted from other types of data by the function as.owin.

They may be manipulated by the functions as.rectangle, as.mask, complement.owin, rotate, shift, affine, erosion, dilation, opening and closing.

Geometrical calculations available for windows include area.owin, perimeter, diameter.owin, boundingbox, eroded.areas, bdist.points, bdist.pixels, and even.breaks.owin. The mapping between continuous coordinates and pixel raster indices is facilitated by the functions raster.x, raster.y and nearest.raster.point.

There is a plot method for window objects, plot.owin. This may be useful if you wish to plot a point pattern's window without the points for graphical purposes.

There are also methods for summary and print.

Warnings

In a window of type "mask", the row index corresponds to increasing y coordinate, and the column index corresponds to increasing x coordinate.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin, as.owin, as.rectangle, as.mask, summary.owin, print.owin, complement.owin, erosion, dilation, opening, closing, affine.owin, shift.owin, rotate.owin, raster.x, raster.y, nearest.raster.point, plot.owin, area.owin, boundingbox, diameter, eroded.areas, bdist.points, bdist.pixels

Examples

```
w <- owin()
w <- owin(c(0,1), c(0,1))
# the unit square

w <- owin(c(0,1), c(0,2))
## Not run:
if(FALSE) {
  plot(w)
  # plots edges of a box 1 unit x 2 units
  v <- locator()
  # click on points in the plot window
  # to be the vertices of a polygon
  # traversed in anticlockwise order
  u <- owin(c(0,1), c(0,2), poly=v)
  plot(u)
  # plots polygonal boundary using polygon()
  plot(as.mask(u, eps=0.02))
  # plots discrete pixel approximation to polygon
}

## End(Not run)
```

---

padimage *Pad the Border of a Pixel Image*

Description

Fills the border of a pixel image with a given value or values, or extends a pixel image to fill a larger window.

Usage

```
padimage(X, value=NA, n=1, W=NULL)
```

Arguments

X	Pixel image (object of class "im").
value	Single value to be placed around the border of X.
n	Width of border, in pixels. See Details.
W	Window for the resulting image. Incompatible with n.

Details

The image X will be expanded by a margin of n pixels, or extended to fill the window W, with new pixel values set to value.

The argument value should be a single value (a vector of length 1), normally a value of the same type as the pixel values of X. It may be NA. Alternatively if X is a factor-valued image, value can be one of the levels of X.

If n is given, it may be a single number, specifying the width of the border in pixels. Alternatively it may be a vector of length 2 or 4. It will be replicated to length 4, and these numbers will be interpreted as the border widths for the (left, right, top, bottom) margins respectively.

Alternatively if W is given, the image will be extended to the window W.

Value

Another object of class "im", of the same type as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

im

Examples

```
Z <- setcov(owin())
plot(padimage(Z, 1, 10))
```

---

pairdist *Pairwise distances*

Description

Computes the matrix of distances between all pairs of 'things' in a dataset

Usage

```
pairdist(X, ...)
```

Arguments

X	Object specifying the locations of a set of 'things' (such as a set of points or a set of line segments).
...	Further arguments depending on the method.

Details

Given a dataset X and Y (representing either a point pattern or a line segment pattern) pairdist computes the distance between each pair of 'things' in the dataset, and returns a matrix containing these distances.

The function pairdist is generic, with methods for point patterns (objects of class "ppp"), line segment patterns (objects of class "psp") and a default method. See the documentation for pairdist.ppp, pairdist.psp or pairdist.default for details.

Value

A square matrix whose `[i,j]` entry is the distance between the 'things' numbered `i` and `j`.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

`pairdist.ppp`, `pairdist.psp`, `pairdist.default`, `crossdist`, `nndist`, `Kest`

`pairdist.default` *Pairwise distances*

Description

Computes the matrix of distances between all pairs of points in a set of points

Usage

```
  ## Default S3 method:
pairdist(X, Y=NULL, ..., period=NULL, method="C", squared=FALSE)
```

Arguments

`X,Y`	Arguments specifying the coordinates of a set of points. Typically `X` and `Y` would be numeric vectors of equal length. Alternatively `Y` may be omitted and `X` may be a list with two components `x` and `y`, or a matrix with two columns.
`...`	Ignored.
`period`	Optional. Dimensions for periodic edge correction.
`method`	String specifying which method of calculation to use. Values are `"C"` and `"interpreted"`. Usually not specified.
`squared`	Logical. If `squared=TRUE`, the squared distances are returned instead (this computation is faster).

Details

Given the coordinates of a set of points, this function computes the Euclidean distances between all pairs of points, and returns the matrix of distances. It is a method for the generic function `pairdist`.

The arguments `X` and `Y` must determine the coordinates of a set of points. Typically `X` and `Y` would be numeric vectors of equal length. Alternatively `Y` may be omitted and `X` may be a list with two components named `x` and `y`, or a matrix or data frame with two columns.

Alternatively if `period` is given, then the distances will be computed in the 'periodic' sense (also known as 'torus' distance). The points will be treated as if they are in a rectangle of width `period[1]` and height `period[2]`. Opposite edges of the rectangle are regarded as equivalent.

If `squared=TRUE` then the *squared* Euclidean distances d^2 are returned, instead of the Euclidean distances d. The squared distances are faster to calculate, and are sufficient for many purposes (such as finding the nearest neighbour of a point).

The argument `method` is not normally used. It is retained only for checking the validity of the software. If `method = "interpreted"` then the distances are computed using interpreted R code only. If `method="C"` (the default) then C code is used. The C code is somewhat faster.

Value

A square matrix whose [i,j] entry is the distance between the points numbered i and j.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

crossdist, nndist, Kest

Examples

```
x <- runif(100)
y <- runif(100)
d <- pairdist(x, y)
d <- pairdist(cbind(x,y))
d <- pairdist(x, y, period=c(1,1))
d <- pairdist(x, y, squared=TRUE)
```

---

pairdist.lpp *Pairwise shortest-path distances between points on a linear network*

Description

Given a pattern of points on a linear network, compute the matrix of distances between all pairs of points, measuring distance by the shortest path in the network.

Usage

```
## S3 method for class 'lpp'
pairdist(X, ..., method="C")
```

Arguments

X	Point pattern on linear network (object of class "lpp").
method	Optional string determining the method of calculation. Either "interpreted" or "C".
...	Ignored.

Details

Given a pattern of points on a linear network, this function computes the matrix of distances between all pairs of points, measuring distance by the shortest path in the network.

If method="C" the distances are computed using code in the C language. If method="interpreted" then the computation is performed using interpreted R code. The R code is much slower, but is provided for checking purposes.

If two points cannot be joined by a path, the distance between them is infinite (Inf).

Value

A symmetric matrix, whose values are nonnegative numbers or infinity (Inf).

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

lpp

Examples

```
X <- runiflpp(12, simplenet)
pairdist(X)
```

---

pairdist.pp3 *Pairwise distances in Three Dimensions*

Description

Computes the matrix of distances between all pairs of points in a three-dimensional point pattern.

Usage

```
  ## S3 method for class 'pp3'
pairdist(X, ..., periodic=FALSE, squared=FALSE)
```

Arguments

X	A point pattern (object of class "pp3").
...	Ignored.
periodic	Logical. Specifies whether to apply a periodic edge correction.
squared	Logical. If squared=TRUE, the squared distances are returned instead (this computation is faster).

Details

This is a method for the generic function pairdist.

Given a three-dimensional point pattern X (an object of class "pp3"), this function computes the Euclidean distances between all pairs of points in X, and returns the matrix of distances.

Alternatively if periodic=TRUE and the window containing X is a box, then the distances will be computed in the 'periodic' sense (also known as 'torus' distance): opposite faces of the box are regarded as equivalent. This is meaningless if the window is not a box.

If squared=TRUE then the *squared* Euclidean distances d^2 are returned, instead of the Euclidean distances d. The squared distances are faster to calculate, and are sufficient for many purposes (such as finding the nearest neighbour of a point).

Value

A square matrix whose [i,j] entry is the distance between the points numbered i and j.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
based on two-dimensional code by Pavel Grabarnik.

See Also

pairdist, crossdist, nndist, K3est

Examples

```
X <- runifpoint3(20)
d <- pairdist(X)
d <- pairdist(X, periodic=TRUE)
d <- pairdist(X, squared=TRUE)
```

---

pairdist.ppp *Pairwise distances*

Description

Computes the matrix of distances between all pairs of points in a point pattern.

Usage

```
## S3 method for class 'ppp'
pairdist(X, ..., periodic=FALSE, method="C", squared=FALSE)
```

Arguments

X	A point pattern (object of class "ppp").
...	Ignored.
periodic	Logical. Specifies whether to apply a periodic edge correction.
method	String specifying which method of calculation to use. Values are "C" and "interpreted". Usually not specified.
squared	Logical. If squared=TRUE, the squared distances are returned instead (this computation is faster).

Details

This is a method for the generic function pairdist.

Given a point pattern X (an object of class "ppp"), this function computes the Euclidean distances between all pairs of points in X, and returns the matrix of distances.

Alternatively if periodic=TRUE and the window containing X is a rectangle, then the distances will be computed in the 'periodic' sense (also known as 'torus' distance): opposite edges of the rectangle are regarded as equivalent. This is meaningless if the window is not a rectangle.

If squared=TRUE then the *squared* Euclidean distances d^2 are returned, instead of the Euclidean distances d. The squared distances are faster to calculate, and are sufficient for many purposes (such as finding the nearest neighbour of a point).

The argument method is not normally used. It is retained only for checking the validity of the software. If method = "interpreted" then the distances are computed using interpreted R code only. If method="C" (the default) then C code is used. The C code is somewhat faster.

Value

A square matrix whose [i,j] entry is the distance between the points numbered i and j.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

pairdist, pairdist.default, pairdist.psp, crossdist, nndist, Kest

Examples

```
data(cells)
d <- pairdist(cells)
d <- pairdist(cells, periodic=TRUE)
d <- pairdist(cells, squared=TRUE)
```

---

pairdist.ppx *Pairwise Distances in Any Dimensions*

Description

Computes the matrix of distances between all pairs of points in a multi-dimensional point pattern.

Usage

```
  ## S3 method for class 'ppx'
pairdist(X, ...)
```

Arguments

X	A point pattern (object of class "ppx").
...	Arguments passed to coords.ppx to determine which coordinates should be used.

Details

This is a method for the generic function pairdist.

Given a multi-dimensional point pattern X (an object of class "ppx"), this function computes the Euclidean distances between all pairs of points in X, and returns the matrix of distances.

By default, both spatial and temporal coordinates are extracted. To obtain the spatial distance between points in a space-time point pattern, set temporal=FALSE.

Value

A square matrix whose [i,j] entry is the distance between the points numbered i and j.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

pairdist, crossdist, nndist

Examples

```
df <- data.frame(x=runif(4),y=runif(4),z=runif(4),w=runif(4))
X <- ppx(data=df)
pairdist(X)
```

---

pairdist.psp *Pairwise distances between line segments*

Description

Computes the matrix of distances between all pairs of line segments in a line segment pattern.

Usage

```
  ## S3 method for class 'psp'
pairdist(X, ..., method="C", type="Hausdorff")
```

Arguments

X	A line segment pattern (object of class "psp").
...	Ignored.
method	String specifying which method of calculation to use. Values are "C" and "interpreted". Usually not specified.
type	Type of distance to be computed. Options are "Hausdorff" and "separation". Partial matching is used.

Details

This function computes the distance between each pair of line segments in X, and returns the matrix of distances.

This is a method for the generic function pairdist for the class "psp".

The distances between line segments are measured in one of two ways:

- if type="Hausdorff", distances are computed in the Hausdorff metric. The Hausdorff distance between two line segments is the *maximum* distance from any point on one of the segments to the nearest point on the other segment.
- if type="separation", distances are computed as the *minimum* distance from a point on one line segment to a point on the other line segment. For example, line segments which cross over each other have separation zero.

The argument method is not normally used. It is retained only for checking the validity of the software. If method = "interpreted" then the distances are computed using interpreted R code only. If method="C" (the default) then compiled C code is used, which is somewhat faster.

Value

A square matrix whose [i,j] entry is the distance between the line segments numbered i and j.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

crossdist, nndist, pairdist.ppp

Examples

```
L <- psp(runif(10), runif(10), runif(10), runif(10), owin())
D <- pairdist(L)
S <- pairdist(L, type="sep")
```

`pairorient` *Point Pair Orientation Distribution*

Description

Computes the distribution of the orientation of vectors joining pairs of points at a particular range of distances.

Usage

```
pairorient(X, r1, r2, ..., cumulative=FALSE,
           correction, ratio = FALSE,
           unit=c("degree", "radian"), domain=NULL)
```

Arguments

`X`	Point pattern (object of class "ppp").
`r1,r2`	Minimum and maximum values of distance to be considered.
`...`	Arguments passed to `circdensity` to control the kernel smoothing, if `cumulative=FALSE`.
`cumulative`	Logical value specifying whether to estimate the probability density (`cumulative=FALSE`, the default) or the cumulative distribution function (`cumulative=TRUE`).
`correction`	Character vector specifying edge correction or corrections. Options are "none", "isotropic", "translate", "good" and "best". Alternatively `correction="all"` selects all options.
`ratio`	Logical. If `TRUE`, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
`unit`	Unit in which the angles should be expressed. Either "degree" or "radian".
`domain`	Optional window. The first point x_i of each pair of points will be constrained to lie in `domain`.

Details

This algorithm considers all pairs of points in the pattern `X` that lie more than `r1` and less than `r2` units apart. The *direction* of the arrow joining the points is measured, as an angle in degrees or radians, anticlockwise from the x axis.

If `cumulative=FALSE` (the default), a kernel estimate of the probability density of the orientations is calculated using `circdensity`.

If `cumulative=TRUE`, then the cumulative distribution function of these directions is calculated. This is the function $O_{r1,r2}(\phi)$ defined in Stoyan and Stoyan (1994), equation (14.53), page 271.

In either case the result can be plotted as a rose diagram by `rose`, or as a function plot by `plot.fv`.

The algorithm gives each observed direction a weight, determined by an edge correction, to adjust for the fact that some interpoint distances are more likely to be observed than others. The choice of edge correction or corrections is determined by the argument `correction`.

It is also possible to calculate an estimate of the probability density from the cumulative distribution function, by numerical differentiation. Use `deriv.fv` with the argument `Dperiodic=TRUE`.

Value

A function value table (object of class `"fv"`) containing the estimates of the probability density or the cumulative distribution function of angles, in degrees (if `unit="degree"`) or radians (if `unit="radian"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

`Kest`, `Ksector`, `nnorient`

Examples

```
rose(pairorient(redwood, 0.05, 0.15, sigma=8), col="grey")
plot(CDF <- pairorient(redwood, 0.05, 0.15, cumulative=TRUE))
plot(f <- deriv(CDF, spar=0.6, Dperiodic=TRUE))
```

PairPiece *The Piecewise Constant Pairwise Interaction Point Process Model*

Description

Creates an instance of a pairwise interaction point process model with piecewise constant potential function. The model can then be fitted to point pattern data.

Usage

```
PairPiece(r)
```

Arguments

r	vector of jump points for the potential function

Details

A pairwise interaction point process in a bounded region is a stochastic point process with probability density of the form

$$f(x_1, \ldots, x_n) = \alpha \prod_i b(x_i) \prod_{i<j} h(x_i, x_j)$$

where $x_1, \ldots, x_n$ represent the points of the pattern. The first product on the right hand side is over all points of the pattern; the second product is over all unordered pairs of points of the pattern.

Thus each point x_i of the pattern contributes a factor $b(x_i)$ to the probability density, and each pair of points x_i, x_j contributes a factor $h(x_i, x_j)$ to the density.

The pairwise interaction term $h(u, v)$ is called *piecewise constant* if it depends only on the distance between u and v, say $h(u, v) = H(||u - v||)$, and H is a piecewise constant function (a function which is constant except for jumps at a finite number of places). The use of piecewise constant interaction terms was first suggested by Takacs (1986).

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the piecewise constant pairwise interaction is yielded by the function `PairPiece()`. See the examples below.

The entries of `r` must be strictly increasing, positive numbers. They are interpreted as the points of discontinuity of H. It is assumed that $H(s) = 1$ for all $s > r_{max}$ where r_{max} is the maximum value in `r`. Thus the model has as many regular parameters (see `ppm`) as there are entries in `r`. The i-th regular parameter θ_i is the logarithm of the value of the interaction function H on the interval $[r_{i-1}, r_i)$.

If `r` is a single number, this model is similar to the Strauss process, see `Strauss`. The difference is that in `PairPiece` the interaction function is continuous on the right, while in `Strauss` it is continuous on the left.

The analogue of this model for multitype point processes has not yet been implemented.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process. The process is a pairwise interaction process, whose interaction potential is piecewise constant, with jumps at the distances given in the vector r.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Takacs, R. (1986) Estimator for the pair potential of a Gibbsian point process. *Statistics* **17**, 429–433.

See Also

ppm, pairwise.family, ppm.object, Strauss rmh.ppm

Examples

```
   PairPiece(c(0.1,0.2))
   # prints a sensible description of itself
   data(cells)

   ## Not run:
   ppm(cells, ~1, PairPiece(r = c(0.05, 0.1, 0.2)))
   # fit a stationary piecewise constant pairwise interaction process

## End(Not run)

   ppm(cells, ~polynom(x,y,3), PairPiece(c(0.05, 0.1)))
   # nonstationary process with log-cubic polynomial trend
```

---

pairs.im *Scatterplot Matrix for Pixel Images*

Description

Produces a scatterplot matrix of the pixel values in two or more pixel images.

Usage

```
## S3 method for class 'im'
pairs(..., plot=TRUE)
```

Arguments

...	Any number of arguments, each of which is either a pixel image (object of class "im") or a named argument to be passed to pairs.default.
plot	Logical. If TRUE, the scatterplot matrix is plotted.

Details

This is a method for the generic function `pairs` for the class of pixel images.

It produces a square array of plot panels, in which each panel shows a scatterplot of the pixel values of one image against the corresponding pixel values of another image.

At least two of the arguments `...` should be pixel images (objects of class `"im"`). Their spatial domains must overlap, but need not have the same pixel dimensions.

First the pixel image domains are intersected, and converted to a common pixel resolution. Then the corresponding pixel values of each image are extracted. Then `pairs.default` is called to plot the scatterplot matrix.

Any arguments in `...` which are not pixel images will be passed to `pairs.default` to control the plot.

Value

Invisible. A `data.frame` containing the corresponding pixel values for each image. The return value also belongs to the class `plotpairsim` which has a plot method, so that it can be re-plotted.

Image or Contour Plots

Since the scatterplots may show very dense concentrations of points, it may be useful to set `panel=panel.image` or `panel=panel.contour` to draw a colour image or contour plot of the kernel-smoothed density of the scatterplot in each panel. The argument `panel` is passed to `pairs.default`. See the help for `panel.image` and `panel.contour`.

Low Level Control of Graphics

To control the appearance of the individual scatterplot panels, see `pairs.default`, `points` or `par`. To control the plotting symbol for the points in the scatterplot, use the arguments `pch`, `col`, `bg` as described under `points` (because the default panel plotter is the function `points`). To suppress the tick marks on the plot axes, type `par(xaxt="n", yaxt="n")` before calling `pairs`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`pairs`, `pairs.default`, `panel.contour`, `panel.image`, `plot.im`, `im`, `par`

Examples

```
X <- density(rpoispp(30))
Y <- density(rpoispp(40))
Z <- density(rpoispp(30))
pairs(X,Y,Z)
```

`pairs.linim` *Scatterplot Matrix for Pixel Images on a Linear Network*

Description

Produces a scatterplot matrix of the pixel values in two or more pixel images on a linear network.

Usage

```
## S3 method for class 'linim'
pairs(..., plot=TRUE, eps=NULL)
```

Arguments

`...`	Any number of arguments, each of which is either a pixel image on a linear network (object of class `"linim"`), a pixel image (object of class `"im"`), or a named argument to be passed to `pairs.default`.
`plot`	Logical. If `TRUE`, the scatterplot matrix is plotted.
`eps`	Optional. Spacing between sample points on the network. A positive number.

Details

This is a method for the generic function `pairs` for the class of pixel images on a linear network.

It produces a square array of plot panels, in which each panel shows a scatterplot of the pixel values of one image against the corresponding pixel values of another image.

At least two of the arguments `...` should be a pixel image on a linear network (object of class `"linim"`). They should be defined on the **same** linear network, but may have different pixel resolutions.

First the pixel values of each image are extracted at a set of sample points equally-spaced across the network. Then `pairs.default` is called to plot the scatterplot matrix.

Any arguments in `...` which are not pixel images will be passed to `pairs.default` to control the plot.

Value

Invisible. A `data.frame` containing the corresponding pixel values for each image. The return value also belongs to the class `plotpairsim` which has a plot method, so that it can be re-plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`pairs.default`, `pairs.im`

Examples

```
fit <- lppm(chicago ~ marks * (x+y))
lam <- predict(fit)
do.call(pairs, lam)
```

`pairsat.family` *Saturated Pairwise Interaction Point Process Family*

Description

An object describing the Saturated Pairwise Interaction family of point process models

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the "saturated pairwise interaction" family of point process models.

If you need to create a specific interaction model for use in spatial pattern analysis, use the function `Saturated()` or the two existing implementations of models in this family, `Geyer()` and `SatPiece()`.

Geyer (1999) introduced the "saturation process", a modification of the Strauss process in which the total contribution to the potential from each point (from its pairwise interaction with all other points) is trimmed to a maximum value c. This model is implemented in the function `Geyer()`.

The present class `pairsat.family` is the extension of this saturation idea to all pairwise interactions. Note that the resulting models are no longer pairwise interaction processes - they have interactions of infinite order.

`pairsat.family` is an object of class `"isf"` containing a function `pairwise$eval` for evaluating the sufficient statistics of any saturated pairwise interaction point process model in which the original pair potentials take an exponential family form.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

`Geyer` to create the Geyer saturation process.

`SatPiece` to create a saturated process with piecewise constant pair potential.

`Saturated` to create a more general saturation model.

Other families: `inforder.family`, `ord.family`, `pairwise.family`.

`Pairwise` *Generic Pairwise Interaction model*

Description

Creates an instance of a pairwise interaction point process model which can then be fitted to point pattern data.

Usage

```
Pairwise(pot, name, par, parnames, printfun)
```

Arguments

`pot`	An R language function giving the user-supplied pairwise interaction potential.
`name`	Character string.
`par`	List of numerical values for irregular parameters
`parnames`	Vector of names of irregular parameters
`printfun`	Do not specify this argument: for internal use only.

Details

This code constructs a member of the pairwise interaction family `pairwise.family` with arbitrary pairwise interaction potential given by the user.

Each pair of points in the point pattern contributes a factor $h(d)$ to the probability density, where d is the distance between the two points. The factor term $h(d)$ is

$$h(d) = \exp(-\theta \mathrm{pot}(d))$$

provided $\mathrm{pot}(d)$ is finite, where θ is the coefficient vector in the model.

The function `pot` must take as its first argument a matrix of interpoint distances, and evaluate the potential for each of these distances. The result must be either a matrix with the same dimensions as its input, or an array with its first two dimensions the same as its input (the latter case corresponds to a vector-valued potential).

If irregular parameters are present, then the second argument to `pot` should be a vector of the same type as `par` giving those parameter values.

The values returned by `pot` may be finite numeric values, or `-Inf` indicating a hard core (that is, the corresponding interpoint distance is forbidden). We define $h(d) = 0$ if $\mathrm{pot}(d) = -\infty$. Thus, a potential value of minus infinity is *always* interpreted as corresponding to $h(d) = 0$, regardless of the sign and magnitude of θ.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

ppm, pairwise.family, ppm.object

Examples

```
#This is the same as StraussHard(r=0.7,h=0.05)
strpot <- function(d,par) {
      r <- par$r
      h <- par$h
      value <- (d <= r)
      value[d < h] <- -Inf
      value
}
mySH <- Pairwise(strpot, "StraussHard process", list(r=0.7,h=0.05),
        c("interaction distance r", "hard core distance h"))
data(cells)
ppm(cells, ~ 1, mySH, correction="isotropic")

# Fiksel (1984) double exponential interaction
# see Stoyan, Kendall, Mecke 1987 p 161

fikspot <- function(d, par) {
   r <- par$r
   h <- par$h
   zeta <- par$zeta
   value <- exp(-zeta * d)
   value[d < h] <- -Inf
   value[d > r] <- 0
   value
}
Fiksel <- Pairwise(fikspot, "Fiksel double exponential process",
                   list(r=3.5, h=1, zeta=1),
                   c("interaction distance r",
                     "hard core distance h",
                     "exponential coefficient zeta"))
data(spruces)
fit <- ppm(unmark(spruces), ~1, Fiksel, rbord=3.5)
fit
plot(fitin(fit), xlim=c(0,4))
coef(fit)
# corresponding values obtained by Fiksel (1984) were -1.9 and -6.0
```

---

pairwise.family *Pairwise Interaction Process Family*

Description

An object describing the family of all pairwise interaction Gibbs point processes.

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the pairwise interaction family of point process models.

If you need to create a specific pairwise interaction model for use in modelling, use the function `Pairwise` or one of the existing functions listed below.

Anyway, `pairwise.family` is an object of class `"isf"` containing a function `pairwise.family$eval` for evaluating the sufficient statistics of any pairwise interaction point process model taking an exponential family form.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Other families: `pairsat.family`, `ord.family`, `inorder.family`.

Pairwise interactions: `Poisson`, `Pairwise`, `PairPiece`, `Fiksel`, `Hardcore`, `LennardJones`, `MultiHard`, `MultiStrauss`, `MultiStraussHard`, `Strauss`, `StraussHard`, `Softcore`.

Other interactions: `AreaInter`, `Geyer`, `Saturated`, `Ord`, `OrdThresh`.

`panel.contour` *Panel Plots using Colour Image or Contour Lines*

Description

These functions can be passed to `pairs` or `coplot` to determine what kind of plotting is done in each panel of a multi-panel graphical display.

Usage

```
panel.contour(x, y, ..., sigma = NULL)

panel.image(x, y, ..., sigma = NULL)

panel.histogram(x, ...)
```

Arguments

`x,y`	Coordinates of points in a scatterplot.
`...`	Extra graphics arguments, passed to `contour.im`, `plot.im` or `rect`, respectively, to control the appearance of the panel.
`sigma`	Bandwidth of kernel smoother, on a scale where x and y range between 0 and 1.

Details

These functions can serve as one of the arguments `panel`, `lower.panel`, `upper.panel`, `diag.panel` passed to graphics commands like `pairs` or `coplot`, to determine what kind of plotting is done in each panel of a multi-panel graphical display. In particular they work with `pairs.im`.

The functions `panel.contour` and `panel.contour` are suitable for the off-diagonal plots which involve two datasets `x` and `y`. They first rescale `x` and `y` to the unit square, then apply kernel smoothing with bandwidth `sigma` using `density.ppp`. Then `panel.contour` draws a contour plot while `panel.image` draws a colour image.

The function `panel.histogram` is suitable for the diagonal plots which involve a single dataset `x`. It displays a histogram of the data.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

pairs.im, pairs.default, panel.smooth

Examples

```
with(bei.extra,
       pairs(grad, elev,
             panel      = panel.contour,
             diag.panel = panel.histogram))
```

---

parameters *Extract Model Parameters in Understandable Form*

Description

Given a fitted model of some kind, this function extracts all the parameters needed to specify the model, and returns them as a list.

Usage

```
parameters(model, ...)

## S3 method for class 'dppm'
parameters(model, ...)

## S3 method for class 'kppm'
parameters(model, ...)

## S3 method for class 'ppm'
parameters(model, ...)

## S3 method for class 'profilepl'
parameters(model, ...)

## S3 method for class 'fii'
parameters(model, ...)

## S3 method for class 'interact'
parameters(model, ...)
```

Arguments

`model`	A fitted model of some kind.
`...`	Arguments passed to methods.

Details

The argument `model` should be a fitted model of some kind. This function extracts all the parameters that would be needed to specify the model, and returns them as a list.

The function `parameters` is generic, with methods for class `"ppm"`, `"kppm"`, `"dppm"` and `"profilepl"` and other classes.

Value

A named list, whose format depends on the fitted model.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`coef`

Examples

```
fit1 <- ppm(cells ~ x, Strauss(0.1))
parameters(fit1)
fit2 <- kppm(redwood ~ x, "Thomas")
parameters(fit2)
```

`parres` *Partial Residuals for Point Process Model*

Description

Computes the smoothed partial residuals, a diagnostic for transformation of a covariate in a Poisson point process model.

Usage

```
parres(model, covariate, ...,
       smooth.effect=FALSE, subregion=NULL,
       bw = "nrd0", adjust=1, from = NULL, to = NULL, n = 512,
       bw.input = c("points", "quad"), bw.restrict=FALSE, covname)
```

Arguments

`model`	Fitted point process model (object of class "ppm").
`covariate`	The covariate of interest. Either a character string matching the name of one of the canonical covariates in the model, or one of the names "x" or "y" referring to the Cartesian coordinates, or one of the names of the covariates given when `model` was fitted, or a pixel image (object of class "im") or `function(x,y)` supplying the values of a covariate at any location.
`smooth.effect`	Logical. Determines the choice of algorithm. See Details.
`subregion`	Optional. A window (object of class "owin") specifying a subset of the spatial domain of the data. The calculation will be confined to the data in this subregion.
`bw`	Smoothing bandwidth or bandwidth rule (passed to `density.default`).
`adjust`	Smoothing bandwidth adjustment factor (passed to `density.default`).
`n, from, to`	Arguments passed to `density.default` to control the number and range of values at which the function will be estimated.
`...`	Additional arguments passed to `density.default`.
`bw.input`	Character string specifying the input data used for automatic bandwidth selection.
`bw.restrict`	Logical value, specifying whether bandwidth selection is performed using data from the entire spatial domain or from the `subregion`.
`covname`	Optional. Character string to use as the name of the covariate.

Details

This command computes the smoothed partial residual diagnostic (Baddeley, Chang, Song and Turner, 2012) for the transformation of a covariate in a Poisson point process model.

The argument `model` must be a fitted Poisson point process model.

The diagnostic works in two different ways:

Canonical covariate: The argument `covariate` may be a character string which is the name of one of the *canonical covariates* in the model. The canonical covariates are the functions Z_j that appear in the expression for the Poisson point process intensity

$$\lambda(u) = \exp(\beta_1 Z_1(u) + \ldots + \beta_p Z_p(u))$$

at spatial location u. Type `names(coef(model))` to see the names of the canonical covariates in `model`. If the selected covariate is Z_j, then the diagnostic plot concerns the model term $\beta_j Z_j(u)$. The plot shows a smooth estimate of a function $h(z)$ that should replace this linear term, that is, $\beta_j Z_j(u)$ should be replaced by $h(Z_j(u))$. The linear function is also plotted as a dotted line.

New covariate: If the argument `covariate` is a pixel image (object of class "im") or a `function(x,y)`, it is assumed to provide the values of a covariate that is not present in the model. Alternatively `covariate` can be the name of a covariate that was supplied when the model was fitted (i.e. in the call to `ppm`) but which does not feature in the model formula. In either case we speak of a new covariate $Z(u)$. If the fitted model intensity is $\lambda(u)$ then we consider modifying this to $\lambda(u) \exp(h(Z(u)))$ where $h(z)$ is some function. The diagnostic plot shows an estimate of $h(z)$. **Warning: in this case the diagnostic is not theoretically justified. This option is provided for research purposes.**

Alternatively covariate can be one of the character strings "x" or "y" signifying the Cartesian coordinates. The behaviour here depends on whether the coordinate was one of the canonical covariates in the model.

If there is more than one canonical covariate in the model that depends on the specified covariate, then the covariate effect is computed using all these canonical covariates. For example in a log-quadratic model which includes the terms x and I(x^2), the quadratic effect involving both these terms will be computed.

There are two choices for the algorithm. If smooth.effect=TRUE, the fitted covariate effect (according to model) is added to the point process residuals, then smoothing is applied to these values. If smooth.effect=FALSE, the point process residuals are smoothed first, and then the fitted covariate effect is added to the result.

The smoothing bandwidth is controlled by the arguments bw, adjust, bw.input and bw.restrict. If bw is a numeric value, then the bandwidth is taken to be adjust * bw. If bw is a string representing a bandwidth selection rule (recognised by density.default) then the bandwidth is selected by this rule.

The data used for automatic bandwidth selection are specified by bw.input and bw.restrict. If bw.input="points" (the default) then bandwidth selection is based on the covariate values at the points of the original point pattern dataset to which the model was fitted. If bw.input="quad" then bandwidth selection is based on the covariate values at every quadrature point used to fit the model. If bw.restrict=TRUE then the bandwidth selection is performed using only data from inside the subregion.

Value

A function value table (object of class "fv") containing the values of the smoothed partial residual, the estimated variance, and the fitted effect of the covariate. Also belongs to the class "parres" which has methods for print and plot.

Slow computation

In a large dataset, computation can be very slow if the default settings are used, because the smoothing bandwidth is selected automatically. To avoid this, specify a numerical value for the bandwidth bw. One strategy is to use a coarser subset of the data to select bw automatically. The selected bandwidth can be read off the print output for parres.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz> , Ya-Mei Chang and Yong Song.

References

Baddeley, A., Chang, Y.-M., Song, Y. and Turner, R. (2013) Residual diagnostics for covariate effects in spatial point process models. *Journal of Computational and Graphical Statistics*, **22**, 886–905.

See Also

addvar, rhohat, rho2hat

Examples

```
X <-  rpoispp(function(x,y){exp(3+x+2*x^2)})
model <- ppm(X, ~x+y)
tra <- parres(model, "x")
plot(tra)
plot(parres(model, "x", subregion=square(0.5)))
model2 <- ppm(X, ~x+I(x^2)+y)
plot(parres(model2, "x"))
Z <- setcov(owin())
plot(parres(model2, Z))
```

---

pcf *Pair Correlation Function*

Description

Estimate the pair correlation function.

Usage

```
pcf(X, ...)
```

Arguments

X	Either the observed data point pattern, or an estimate of its K function, or an array of multitype K functions (see Details).
...	Other arguments passed to the appropriate method.

Details

The pair correlation function of a stationary point process is

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$, the reduced second moment function (aka "Ripley's K function") of the point process. See Kest for information about $K(r)$. For a stationary Poisson process, the pair correlation function is identically equal to 1. Values $g(r) < 1$ suggest inhibition between points; values greater than 1 suggest clustering.

We also apply the same definition to other variants of the classical K function, such as the multitype K functions (see Kcross, Kdot) and the inhomogeneous K function (see Kinhom). For all these variants, the benchmark value of $K(r) = \pi r^2$ corresponds to $g(r) = 1$.

This routine computes an estimate of $g(r)$ either directly from a point pattern, or indirectly from an estimate of $K(r)$ or one of its variants.

This function is generic, with methods for the classes "ppp", "fv" and "fasp".

If X is a point pattern (object of class "ppp") then the pair correlation function is estimated using a traditional kernel smoothing method (Stoyan and Stoyan, 1994). See pcf.ppp for details.

If X is a function value table (object of class "fv"), then it is assumed to contain estimates of the K function or one of its variants (typically obtained from Kest or Kinhom). This routine computes an estimate of $g(r)$ using smoothing splines to approximate the derivative. See pcf.fv for details.

If X is a function value array (object of class "fasp"), then it is assumed to contain estimates of several K functions (typically obtained from Kmulti or alltypes). This routine computes an estimate of $g(r)$ for each cell in the array, using smoothing splines to approximate the derivatives. See pcf.fasp for details.

Value

Either a function value table (object of class "fv", see fv.object) representing a pair correlation function, or a function array (object of class "fasp", see fasp.object) representing an array of pair correlation functions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

References

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

pcf.ppp, pcf.fv, pcf.fasp, Kest, Kinhom, Kcross, Kdot, Kmulti, alltypes

Examples

```
# ppp object
X <- simdat

p <- pcf(X)
plot(p)

# fv object
K <- Kest(X)
p2 <- pcf(K, spar=0.8, method="b")
plot(p2)

# multitype pattern; fasp object
amaK <- alltypes(amacrine, "K")
amap <- pcf(amaK, spar=1, method="b")
plot(amap)
```

---

pcf.fasp *Pair Correlation Function obtained from array of K functions*

Description

Estimates the (bivariate) pair correlation functions of a point pattern, given an array of (bivariate) K functions.

Usage

```
## S3 method for class 'fasp'
pcf(X, ..., method="c")
```

Arguments

X	An array of multitype K functions (object of class "fasp").
...	Arguments controlling the smoothing spline function smooth.spline.
method	Letter "a", "b", "c" or "d" indicating the method for deriving the pair correlation function from the K function.

Details

The pair correlation function of a stationary point process is

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$, the reduced second moment function (aka "Ripley's K function") of the point process. See Kest for information about $K(r)$. For a stationary Poisson process, the pair correlation function is identically equal to 1. Values $g(r) < 1$ suggest inhibition between points; values greater than 1 suggest clustering.

We also apply the same definition to other variants of the classical K function, such as the multitype K functions (see Kcross, Kdot) and the inhomogeneous K function (see Kinhom). For all these variants, the benchmark value of $K(r) = \pi r^2$ corresponds to $g(r) = 1$.

This routine computes an estimate of $g(r)$ from an array of estimates of $K(r)$ or its variants, using smoothing splines to approximate the derivatives. It is a method for the generic function pcf.

The argument X should be a function array (object of class "fasp", see fasp.object) containing several estimates of K functions. This should have been obtained from alltypes with the argument fun="K".

The smoothing spline operations are performed by smooth.spline and predict.smooth.spline from the modreg library. Four numerical methods are available:

- **"a"** apply smoothing to $K(r)$, estimate its derivative, and plug in to the formula above;
- **"b"** apply smoothing to $Y(r) = \frac{K(r)}{2\pi r}$ constraining $Y(0) = 0$, estimate the derivative of Y, and solve;
- **"c"** apply smoothing to $Z(r) = \frac{K(r)}{\pi r^2}$ constraining $Z(0) = 1$, estimate its derivative, and solve.
- **"d"** apply smoothing to $V(r) = \sqrt{K(r)}$, estimate its derivative, and solve.

Method "c" seems to be the best at suppressing variability for small values of r. However it effectively constrains $g(0) = 1$. If the point pattern seems to have inhibition at small distances, you may wish to experiment with method "b" which effectively constrains $g(0) = 0$. Method "a" seems comparatively unreliable.

Useful arguments to control the splines include the smoothing tradeoff parameter spar and the degrees of freedom df. See smooth.spline for details.

Value

A function array (object of class "fasp", see fasp.object) representing an array of pair correlation functions. This can be thought of as a matrix Y each of whose entries Y[i,j] is a function value table (class "fv") representing the pair correlation function between points of type i and points of type j.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

Kest, Kinhom, Kcross, Kdot, Kmulti, alltypes, smooth.spline, predict.smooth.spline

Examples

```
# multitype point pattern
KK <- alltypes(amacrine, "K")
p <- pcf.fasp(KK, spar=0.5, method="b")
plot(p)
# strong inhibition between points of the same type
```

---

pcf.fv *Pair Correlation Function obtained from K Function*

Description

Estimates the pair correlation function of a point pattern, given an estimate of the K function.

Usage

```
  ## S3 method for class 'fv'
pcf(X, ..., method="c")
```

Arguments

X	An estimate of the K function or one of its variants. An object of class "fv".
...	Arguments controlling the smoothing spline function smooth.spline.
method	Letter "a", "b", "c" or "d" indicating the method for deriving the pair correlation function from the K function.

Details

The pair correlation function of a stationary point process is

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$, the reduced second moment function (aka “Ripley's K function”) of the point process. See Kest for information about $K(r)$. For a stationary Poisson process, the pair correlation function is identically equal to 1. Values $g(r) < 1$ suggest inhibition between points; values greater than 1 suggest clustering.

We also apply the same definition to other variants of the classical K function, such as the multitype K functions (see `Kcross`, `Kdot`) and the inhomogeneous K function (see `Kinhom`). For all these variants, the benchmark value of $K(r) = \pi r^2$ corresponds to $g(r) = 1$.

This routine computes an estimate of $g(r)$ from an estimate of $K(r)$ or its variants, using smoothing splines to approximate the derivative. It is a method for the generic function `pcf` for the class `"fv"`.

The argument `X` should be an estimated K function, given as a function value table (object of class `"fv"`, see `fv.object`). This object should be the value returned by `Kest`, `Kcross`, `Kmulti` or `Kinhom`.

The smoothing spline operations are performed by `smooth.spline` and `predict.smooth.spline` from the `modreg` library. Four numerical methods are available:

- **"a"** apply smoothing to $K(r)$, estimate its derivative, and plug in to the formula above;
- **"b"** apply smoothing to $Y(r) = \frac{K(r)}{2\pi r}$ constraining $Y(0) = 0$, estimate the derivative of Y, and solve;
- **"c"** apply smoothing to $Z(r) = \frac{K(r)}{\pi r^2}$ constraining $Z(0) = 1$, estimate its derivative, and solve.
- **"d"** apply smoothing to $V(r) = \sqrt{K(r)}$, estimate its derivative, and solve.

Method `"c"` seems to be the best at suppressing variability for small values of r. However it effectively constrains $g(0) = 1$. If the point pattern seems to have inhibition at small distances, you may wish to experiment with method `"b"` which effectively constrains $g(0) = 0$. Method `"a"` seems comparatively unreliable.

Useful arguments to control the splines include the smoothing tradeoff parameter `spar` and the degrees of freedom `df`. See `smooth.spline` for details.

Value

A function value table (object of class `"fv"`, see `fv.object`) representing a pair correlation function.

Essentially a data frame containing (at least) the variables

`r`	the vector of values of the argument r at which the pair correlation function $g(r)$ has been estimated
`pcf`	vector of values of $g(r)$

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`r.turner@auckland.ac.nz`> and Ege Rubak <`rubak@math.aau.dk`>.

References

Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

`pcf`, `pcf.ppp`, `Kest`, `Kinhom`, `Kcross`, `Kdot`, `Kmulti`, `alltypes`, `smooth.spline`, `predict.smooth.spline`

Examples

```
# univariate point pattern
X <- simdat

K <- Kest(X)
p <- pcf.fv(K, spar=0.5, method="b")
plot(p, main="pair correlation function for simdat")
# indicates inhibition at distances r < 0.3
```

---

pcf.ppp *Pair Correlation Function of Point Pattern*

Description

Estimates the pair correlation function of a point pattern using kernel methods.

Usage

```
## S3 method for class 'ppp'
pcf(X, ..., r = NULL, kernel="epanechnikov", bw=NULL,
                  stoyan=0.15,
                  correction=c("translate", "Ripley"),
                  divisor = c("r", "d"),
                  var.approx = FALSE,
                  domain=NULL,
                  ratio=FALSE, close=NULL)
```

Arguments

X	A point pattern (object of class "ppp").
r	Vector of values for the argument r at which $g(r)$ should be evaluated. There is a sensible default.
kernel	Choice of smoothing kernel, passed to density.default.
bw	Bandwidth for smoothing kernel, passed to density.default. Either a single numeric value, or a character string specifying a bandwidth selection rule recognised by density.default. If bw is missing or NULL, the default value is computed using Stoyan's rule of thumb: see Details.
...	Other arguments passed to the kernel density estimation function density.default.
stoyan	Coefficient for Stoyan's bandwidth selection rule; see Details.
correction	Choice of edge correction.
divisor	Choice of divisor in the estimation formula: either "r" (the default) or "d". See Details.
var.approx	Logical value indicating whether to compute an analytic approximation to the variance of the estimated pair correlation.
domain	Optional. Calculations will be restricted to this subset of the window. See Details.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
close	Advanced use only. Precomputed data. See section on Advanced Use.

Details

The pair correlation function $g(r)$ is a summary of the dependence between points in a spatial point process. The best intuitive interpretation is the following: the probability $p(r)$ of finding two points at locations x and y separated by a distance r is equal to

$$p(r) = \lambda^2 g(r) \, \mathrm{d}x \, \mathrm{d}y$$

where λ is the intensity of the point process. For a completely random (uniform Poisson) process, $p(r) = \lambda^2 \, \mathrm{d}x \, \mathrm{d}y$ so $g(r) = 1$. Formally, the pair correlation function of a stationary point process is defined by

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$, the reduced second moment function (aka "Ripley's K function") of the point process. See `Kest` for information about $K(r)$.

For a stationary Poisson process, the pair correlation function is identically equal to 1. Values $g(r) < 1$ suggest inhibition between points; values greater than 1 suggest clustering.

This routine computes an estimate of $g(r)$ by kernel smoothing.

- If `divisor="r"` (the default), then the standard kernel estimator (Stoyan and Stoyan, 1994, pages 284–285) is used. By default, the recommendations of Stoyan and Stoyan (1994) are followed exactly.
- If `divisor="d"` then a modified estimator is used: the contribution from an interpoint distance d_{ij} to the estimate of $g(r)$ is divided by d_{ij} instead of dividing by r. This usually improves the bias of the estimator when r is close to zero.

There is also a choice of spatial edge corrections (which are needed to avoid bias due to edge effects associated with the boundary of the spatial window):

- If `correction="translate"` or `correction="translation"` then the translation correction is used. For `divisor="r"` the translation-corrected estimate is given in equation (15.15), page 284 of Stoyan and Stoyan (1994).
- If `correction="Ripley"` then Ripley's isotropic edge correction is used. For `divisor="r"` the isotropic-corrected estimate is given in equation (15.18), page 285 of Stoyan and Stoyan (1994).
- If `correction=c("translate", "Ripley")` then both estimates will be computed.

Alternatively `correction="all"` selects all options.

The choice of smoothing kernel is controlled by the argument `kernel` which is passed to `density.default`. The default is the Epanechnikov kernel, recommended by Stoyan and Stoyan (1994, page 285).

The bandwidth of the smoothing kernel can be controlled by the argument `bw`. Its precise interpretation is explained in the documentation for `density.default`. For the Epanechnikov kernel, the argument `bw` is equivalent to $h/\sqrt{5}$.

Stoyan and Stoyan (1994, page 285) recommend using the Epanechnikov kernel with support $[-h, h]$ chosen by the rule of thumn $h = c/\sqrt{\lambda}$, where λ is the (estimated) intensity of the point process, and c is a constant in the range from 0.1 to 0.2. See equation (15.16). If `bw` is missing or `NULL`, then this rule of thumb will be applied. The argument `stoyan` determines the value of c. The smoothing bandwidth that was used in the calculation is returned as an attribute of the final result.

The argument `r` is the vector of values for the distance r at which $g(r)$ should be evaluated. There is a sensible default. If it is specified, `r` must be a vector of increasing numbers starting from `r[1] = 0`, and `max(r)` must not exceed half the diameter of the window.

If the argument domain is given, estimation will be restricted to this region. That is, the estimate of $g(r)$ will be based on pairs of points in which the first point lies inside domain and the second point is unrestricted. The argument domain should be a window (object of class "owin") or something acceptable to as.owin. It must be a subset of the window of the point pattern X.

To compute a confidence band for the true value of the pair correlation function, use lohboot.

If var.approx = TRUE, the variance of the estimate of the pair correlation will also be calculated using an analytic approximation (Illian et al, 2008, page 234) which is valid for stationary point processes which are not too clustered. This calculation is not yet implemented when the argument domain is given.

Value

A function value table (object of class "fv"). Essentially a data frame containing the variables

r	the vector of values of the argument r at which the pair correlation function $g(r)$ has been estimated
theo	vector of values equal to 1, the theoretical value of $g(r)$ for the Poisson process
trans	vector of values of $g(r)$ estimated by translation correction
iso	vector of values of $g(r)$ estimated by Ripley isotropic correction
v	vector of approximate values of the variance of the estimate of $g(r)$

as required.

If ratio=TRUE then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $g(r)$.

The return value also has an attribute "bw" giving the smoothing bandwidth that was used.

Advanced Use

To perform the same computation using several different bandwidths bw, it is efficient to use the argument close. This should be the result of closepairs(X, rmax) for a suitably large value of rmax, namely rmax >= max(r) + 3 * bw.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> and Martin Hazelton.

References

Illian, J., Penttinen, A., Stoyan, H. and Stoyan, D. (2008) *Statistical Analysis and Modelling of Spatial Point Patterns.* Wiley.

Stoyan, D. and Stoyan, H. (1994) *Fractals, random shapes and point fields: methods of geometrical statistics.* John Wiley and Sons.

See Also

Kest, pcf, density.default, bw.stoyan, bw.pcf, lohboot.

Examples

```
X <- simdat

p <- pcf(X)
plot(p, main="pair correlation function for X")
# indicates inhibition at distances r < 0.3

pd <- pcf(X, divisor="d")

# compare estimates
plot(p, cbind(iso, theo) ~ r, col=c("blue", "red"),
       ylim.covers=0, main="", lwd=c(2,1), lty=c(1,3), legend=FALSE)
plot(pd, iso ~ r, col="green", lwd=2, add=TRUE)
legend("center", col=c("blue", "green"), lty=1, lwd=2,
       legend=c("divisor=r","divisor=d"))

# calculate approximate variance and show POINTWISE confidence bands
pv <- pcf(X, var.approx=TRUE)
plot(pv, cbind(iso, iso+2*sqrt(v), iso-2*sqrt(v)) ~ r)
```

`pcf3est` *Pair Correlation Function of a Three-Dimensional Point Pattern*

Description

Estimates the pair correlation function from a three-dimensional point pattern.

Usage

```
pcf3est(X, ..., rmax = NULL, nrval = 128, correction = c("translation",
"isotropic"), delta=NULL, adjust=1, biascorrect=TRUE)
```

Arguments

`X`	Three-dimensional point pattern (object of class "pp3").
`...`	Ignored.
`rmax`	Optional. Maximum value of argument r for which $g_3(r)$ will be estimated.
`nrval`	Optional. Number of values of r for which $g_3(r)$ will be estimated.
`correction`	Optional. Character vector specifying the edge correction(s) to be applied. See Details.
`delta`	Optional. Half-width of the Epanechnikov smoothing kernel.
`adjust`	Optional. Adjustment factor for the default value of `delta`.
`biascorrect`	Logical value. Whether to correct for underestimation due to truncation of the kernel near $r = 0$.

Details

For a stationary point process Φ in three-dimensional space, the pair correlation function is

$$g_3(r) = \frac{K_3'(r)}{4\pi r^2}$$

where K_3' is the derivative of the three-dimensional K-function (see `K3est`).

The three-dimensional point pattern `X` is assumed to be a partial realisation of a stationary point process Φ. The distance between each pair of distinct points is computed. Kernel smoothing is applied to these distance values (weighted by an edge correction factor) and the result is renormalised to give the estimate of $g_3(r)$.

The available edge corrections are:

`"translation"`: the Ohser translation correction estimator (Ohser, 1983; Baddeley et al, 1993)

`"isotropic"`: the three-dimensional counterpart of Ripley's isotropic edge correction (Ripley, 1977; Baddeley et al, 1993).

Kernel smoothing is performed using the Epanechnikov kernel with half-width `delta`. If `delta` is missing, the default is to use the rule-of-thumb $\delta = 0.26/\lambda^{1/3}$ where $\lambda = n/v$ is the estimated intensity, computed from the number n of data points and the volume v of the enclosing box. This default value of `delta` is multiplied by the factor `adjust`.

The smoothing estimate of the pair correlation $g_3(r)$ is typically an underestimate when r is small, due to truncation of the kernel at $r = 0$. If `biascorrect=TRUE`, the smoothed estimate is approximately adjusted for this bias. This is advisable whenever the dataset contains a sufficiently large number of points.

Value

A function value table (object of class `"fv"`) that can be plotted, printed or coerced to a data frame containing the function values.

Additionally the value of `delta` is returned as an attribute of this object.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rana Moyeed.

References

Baddeley, A.J, Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.

Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

`K3est`, `pcf`

Examples

```
X <- rpoispp3(250)
Z <- pcf3est(X)
Zbias <- pcf3est(X, biascorrect=FALSE)
if(interactive()) {
  opa <- par(mfrow=c(1,2))
  plot(Z,     ylim.covers=c(0, 1.2))
  plot(Zbias, ylim.covers=c(0, 1.2))
  par(opa)
}
attr(Z, "delta")
```

---

pcfcross *Multitype pair correlation function (cross-type)*

Description

Calculates an estimate of the cross-type pair correlation function for a multitype point pattern.

Usage

```
pcfcross(X, i, j, ...,
         r = NULL,
         kernel = "epanechnikov", bw = NULL, stoyan = 0.15,
         correction = c("isotropic", "Ripley", "translate"),
         divisor = c("r", "d"))
```

Arguments

X	The observed point pattern, from which an estimate of the cross-type pair correlation function $g_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
...	Ignored.
r	Vector of values for the argument r at which $g(r)$ should be evaluated. There is a sensible default.
kernel	Choice of smoothing kernel, passed to density.default.
bw	Bandwidth for smoothing kernel, passed to density.default.
stoyan	Coefficient for default bandwidth rule; see Details.
correction	Choice of edge correction.
divisor	Choice of divisor in the estimation formula: either "r" (the default) or "d". See Details.

Details

The cross-type pair correlation function is a generalisation of the pair correlation function pcf to multitype point patterns.

For two locations x and y separated by a distance r, the probability $p(r)$ of finding a point of type i at location x and a point of type j at location y is

$$p(r) = \lambda_i \lambda_j g_{i,j}(r) \,\mathrm{d}x \,\mathrm{d}y$$

where λ_i is the intensity of the points of type i. For a completely random Poisson marked point process, $p(r) = \lambda_i \lambda_j$ so $g_{i,j}(r) = 1$. Indeed for any marked point pattern in which the points of type `i` are independent of the points of type `j`, the theoretical value of the cross-type pair correlation is $g_{i,j}(r) = 1$.

For a stationary multitype point process, the cross-type pair correlation function between marks i and j is formally defined as

$$g_{i,j}(r) = \frac{K'_{i,j}(r)}{2\pi r}$$

where $K'_{i,j}$ is the derivative of the cross-type K function $K_{i,j}(r)$. of the point process. See Kest for information about $K(r)$.

The command `pcfcross` computes a kernel estimate of the cross-type pair correlation function between marks i and j.

- If `divisor="r"` (the default), then the multitype counterpart of the standard kernel estimator (Stoyan and Stoyan, 1994, pages 284–285) is used. By default, the recommendations of Stoyan and Stoyan (1994) are followed exactly.
- If `divisor="d"` then a modified estimator is used: the contribution from an interpoint distance d_{ij} to the estimate of $g(r)$ is divided by d_{ij} instead of dividing by r. This usually improves the bias of the estimator when r is close to zero.

There is also a choice of spatial edge corrections (which are needed to avoid bias due to edge effects associated with the boundary of the spatial window): `correction="translate"` is the Ohser-Stoyan translation correction, and `correction="isotropic"` or `"Ripley"` is Ripley's isotropic correction.

The choice of smoothing kernel is controlled by the argument `kernel` which is passed to density. The default is the Epanechnikov kernel.

The bandwidth of the smoothing kernel can be controlled by the argument `bw`. Its precise interpretation is explained in the documentation for density.default. For the Epanechnikov kernel with support $[-h, h]$, the argument `bw` is equivalent to $h/\sqrt{5}$.

If `bw` is not specified, the default bandwidth is determined by Stoyan's rule of thumb (Stoyan and Stoyan, 1994, page 285) applied to the points of type `j`. That is, $h = c/\sqrt{\lambda}$, where λ is the (estimated) intensity of the point process of type `j`, and c is a constant in the range from 0.1 to 0.2. The argument `stoyan` determines the value of c.

The companion function pcfdot computes the corresponding analogue of Kdot.

Value

An object of class `"fv"`, see fv.object, which can be plotted directly using plot.fv.

Essentially a data frame containing columns

`r` the vector of values of the argument r at which the function $g_{i,j}$ has been estimated theo the theoretical value $g_{i,j}(r) = 1$ for independent marks.

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $g_{i,j}$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Mark connection function markconnect.

Multitype pair correlation pcfdot, pcfmulti.

Pair correlation pcf,pcf.ppp.

Kcross

Examples

```
data(amacrine)
p <- pcfcross(amacrine, "off", "on")
p <- pcfcross(amacrine, "off", "on", stoyan=0.1)
plot(p)
```

---

pcfcross.inhom *Inhomogeneous Multitype Pair Correlation Function (Cross-Type)*

Description

Estimates the inhomogeneous cross-type pair correlation function for a multitype point pattern.

Usage

```
pcfcross.inhom(X, i, j, lambdaI = NULL, lambdaJ = NULL, ...,
               r = NULL, breaks = NULL,
               kernel="epanechnikov", bw=NULL, stoyan=0.15,
               correction = c("isotropic", "Ripley", "translate"),
               sigma = NULL, varcov = NULL)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous cross-type pair correlation function $g_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).

lambdaI	Optional. Values of the estimated intensity function of the points of type i. Either a vector giving the intensity values at the points of type i, a pixel image (object of class "im") giving the intensity values at all locations, or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdaJ	Optional. Values of the estimated intensity function of the points of type j. A numeric vector, pixel image or function(x,y).
r	Vector of values for the argument r at which $g_{ij}(r)$ should be evaluated. There is a sensible default.
breaks	This argument is for internal use only.
kernel	Choice of smoothing kernel, passed to density.default.
bw	Bandwidth for smoothing kernel, passed to density.default.
...	Other arguments passed to the kernel density estimation function density.default.
stoyan	Bandwidth coefficient; see Details.
correction	Choice of edge correction.
sigma,varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambdaI or lambdaJ is estimated by kernel smoothing.

Details

The inhomogeneous cross-type pair correlation function $g_{ij}(r)$ is a summary of the dependence between two types of points in a multitype spatial point process that does not have a uniform density of points.

The best intuitive interpretation is the following: the probability $p(r)$ of finding two points, of types i and j respectively, at locations x and y separated by a distance r is equal to

$$p(r) = \lambda_i(x) lambda_j(y) g(r) \,\mathrm{d}x \,\mathrm{d}y$$

where λ_i is the intensity function of the process of points of type i. For a multitype Poisson point process, this probability is $p(r) = \lambda_i(x)\lambda_j(y)$ so $g_{ij}(r) = 1$.

The command pcfcross.inhom estimates the inhomogeneous pair correlation using a modified version of the algorithm in pcf.ppp.

If the arguments lambdaI and lambdaJ are missing or null, they are estimated from X by kernel smoothing using a leave-one-out estimator.

Value

A function value table (object of class "fv"). Essentially a data frame containing the variables

r	the vector of values of the argument r at which the inhomogeneous cross-type pair correlation function $g_{ij}(r)$ has been estimated
theo	vector of values equal to 1, the theoretical value of $g_{ij}(r)$ for the Poisson process
trans	vector of values of $g_{ij}(r)$ estimated by translation correction
iso	vector of values of $g_{ij}(r)$ estimated by Ripley isotropic correction

as required.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

pcf.ppp, pcfinhom, pcfcross, pcfdot.inhom

Examples

```
data(amacrine)
plot(pcfcross.inhom(amacrine, "on", "off", stoyan=0.1),
     legendpos="bottom")
```

---

pcfdot *Multitype pair correlation function (i-to-any)*

Description

Calculates an estimate of the multitype pair correlation function (from points of type `i` to points of any type) for a multitype point pattern.

Usage

```
pcfdot(X, i, ..., r = NULL,
       kernel = "epanechnikov", bw = NULL, stoyan = 0.15,
       correction = c("isotropic", "Ripley", "translate"),
       divisor = c("r", "d"))
```

Arguments

`X`	The observed point pattern, from which an estimate of the dot-type pair correlation function $g_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).
`i`	The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.
`...`	Ignored.
`r`	Vector of values for the argument r at which $g(r)$ should be evaluated. There is a sensible default.
`kernel`	Choice of smoothing kernel, passed to density.default.
`bw`	Bandwidth for smoothing kernel, passed to density.default.
`stoyan`	Coefficient for default bandwidth rule; see Details.
`correction`	Choice of edge correction.
`divisor`	Choice of divisor in the estimation formula: either `"r"` (the default) or `"d"`. See Details.

Details

This is a generalisation of the pair correlation function `pcf` to multitype point patterns.

For two locations x and y separated by a nonzero distance r, the probability $p(r)$ of finding a point of type i at location x and a point of any type at location y is

$$p(r) = \lambda_i \lambda g_{i\bullet}(r) \,\mathrm{d}x\,\mathrm{d}y$$

where λ is the intensity of all points, and λ_i is the intensity of the points of type i. For a completely random Poisson marked point process, $p(r) = \lambda_i \lambda$ so $g_{i\bullet}(r) = 1$.

For a stationary multitype point process, the type-`i`-to-any-type pair correlation function between marks i and j is formally defined as

$$g_{i\bullet}(r) = \frac{K'_{i\bullet}(r)}{2\pi r}$$

where $K'_{i\bullet}$ is the derivative of the type-`i`-to-any-type K function $K_{i\bullet}(r)$. of the point process. See `Kdot` for information about $K_{i\bullet}(r)$.

The command `pcfdot` computes a kernel estimate of the multitype pair correlation function from points of type i to points of any type.

- If `divisor="r"` (the default), then the multitype counterpart of the standard kernel estimator (Stoyan and Stoyan, 1994, pages 284–285) is used. By default, the recommendations of Stoyan and Stoyan (1994) are followed exactly.
- If `divisor="d"` then a modified estimator is used: the contribution from an interpoint distance d_{ij} to the estimate of $g(r)$ is divided by d_{ij} instead of dividing by r. This usually improves the bias of the estimator when r is close to zero.

There is also a choice of spatial edge corrections (which are needed to avoid bias due to edge effects associated with the boundary of the spatial window): `correction="translate"` is the Ohser-Stoyan translation correction, and `correction="isotropic"` or `"Ripley"` is Ripley's isotropic correction.

The choice of smoothing kernel is controlled by the argument `kernel` which is passed to `density`. The default is the Epanechnikov kernel.

The bandwidth of the smoothing kernel can be controlled by the argument `bw`. Its precise interpretation is explained in the documentation for `density.default`. For the Epanechnikov kernel with support $[-h, h]$, the argument `bw` is equivalent to $h/\sqrt{5}$.

If `bw` is not specified, the default bandwidth is determined by Stoyan's rule of thumb (Stoyan and Stoyan, 1994, page 285). That is, $h = c/\sqrt{\lambda}$, where λ is the (estimated) intensity of the unmarked point process, and c is a constant in the range from 0.1 to 0.2. The argument `stoyan` determines the value of c.

The companion function `pcfcross` computes the corresponding analogue of `Kcross`.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing columns

`r`	the vector of values of the argument r at which the function $g_{i\bullet}$ has been estimated
`theo`	the theoretical value $g_{i\bullet}(r) = 1$ for independent marks.

together with columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $g_{i,j}$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Mark connection function `markconnect`.

Multitype pair correlation `pcfcross`, `pcfmulti`.

Pair correlation `pcf`,`pcf.ppp`.

`Kdot`

Examples

```
data(amacrine)
p <- pcfdot(amacrine, "on")
p <- pcfdot(amacrine, "on", stoyan=0.1)
plot(p)
```

`pcfdot.inhom` *Inhomogeneous Multitype Pair Correlation Function (Type-i-To-Any-Type)*

Description

Estimates the inhomogeneous multitype pair correlation function (from type i to any type) for a multitype point pattern.

Usage

```
pcfdot.inhom(X, i, lambdaI = NULL, lambdadot = NULL, ...,
               r = NULL, breaks = NULL,
               kernel="epanechnikov", bw=NULL, stoyan=0.15,
               correction = c("isotropic", "Ripley", "translate"),
               sigma = NULL, varcov = NULL)
```

Arguments

`X` The observed point pattern, from which an estimate of the inhomogeneous multitype pair correlation function $g_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).

`i` The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.

`lambdaI` Optional. Values of the estimated intensity function of the points of type `i`. Either a vector giving the intensity values at the points of type `i`, a pixel image (object of class `"im"`) giving the intensity values at all locations, or a `function(x,y)` which can be evaluated to give the intensity value at any location.

`lambdadot` Optional. Values of the estimated intensity function of the point pattern `X`. A numeric vector, pixel image or `function(x,y)`.

r	Vector of values for the argument r at which $g_{i\bullet}(r)$ should be evaluated. There is a sensible default.
breaks	This argument is for internal use only.
kernel	Choice of smoothing kernel, passed to density.default.
bw	Bandwidth for smoothing kernel, passed to density.default.
...	Other arguments passed to the kernel density estimation function density.default.
stoyan	Bandwidth coefficient; see Details.
correction	Choice of edge correction.
sigma,varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambdaI or lambdadot is estimated by kernel smoothing.

Details

The inhomogeneous multitype (type i to any type) pair correlation function $g_{i\bullet}(r)$ is a summary of the dependence between different types of points in a multitype spatial point process that does not have a uniform density of points.

The best intuitive interpretation is the following: the probability $p(r)$ of finding a point of type i at location x and another point of any type at location y, where x and y are separated by a distance r, is equal to

$$p(r) = \lambda_i(x) lambda(y) g(r)\, \mathrm{d}x\, \mathrm{d}y$$

where λ_i is the intensity function of the process of points of type i, and where λ is the intensity function of the points of all types. For a multitype Poisson point process, this probability is $p(r) = \lambda_i(x)\lambda(y)$ so $g_{i\bullet}(r) = 1$.

The command `pcfdot.inhom` estimates the inhomogeneous multitype pair correlation using a modified version of the algorithm in pcf.ppp.

If the arguments `lambdaI` and `lambdadot` are missing or null, they are estimated from `X` by kernel smoothing using a leave-one-out estimator.

Value

A function value table (object of class `"fv"`). Essentially a data frame containing the variables

r	the vector of values of the argument r at which the inhomogeneous multitype pair correlation function $g_{i\bullet}(r)$ has been estimated
theo	vector of values equal to 1, the theoretical value of $g_{i\bullet}(r)$ for the Poisson process
trans	vector of values of $g_{i\bullet}(r)$ estimated by translation correction
iso	vector of values of $g_{i\bullet}(r)$ estimated by Ripley isotropic correction

as required.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

pcf.ppp, pcfinhom, pcfdot, pcfcross.inhom

Examples

```
data(amacrine)
plot(pcfdot.inhom(amacrine, "on", stoyan=0.1), legendpos="bottom")
```

---

pcfinhom *Inhomogeneous Pair Correlation Function*

Description

Estimates the inhomogeneous pair correlation function of a point pattern using kernel methods.

Usage

```
pcfinhom(X, lambda = NULL, ..., r = NULL,
         kernel = "epanechnikov", bw = NULL, stoyan = 0.15,
         correction = c("translate", "Ripley"),
         divisor = c("r", "d"),
         renormalise = TRUE, normpower=1,
         update = TRUE, leaveoneout = TRUE,
         reciplambda = NULL,
         sigma = NULL, varcov = NULL, close=NULL)
```

Arguments

X	A point pattern (object of class "ppp").
lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
r	Vector of values for the argument r at which $g(r)$ should be evaluated. There is a sensible default.
kernel	Choice of smoothing kernel, passed to density.default.
bw	Bandwidth for smoothing kernel, passed to density.default. Either a single numeric value, or a character string specifying a bandwidth selection rule recognised by density.default. If bw is missing or NULL, the default value is computed using Stoyan's rule of thumb: see bw.stoyan.
...	Other arguments passed to the kernel density estimation function density.default.
stoyan	Coefficient for Stoyan's bandwidth selection rule; see bw.stoyan.
correction	Choice of edge correction.
divisor	Choice of divisor in the estimation formula: either "r" (the default) or "d". See pcf.ppp.
renormalise	Logical. Whether to renormalise the estimate. See Details.
normpower	Integer (usually either 1 or 2). Normalisation power. See Details.
update	Logical. If lambda is a fitted model (class "ppm", "kppm" or "dppm") and update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without re-fitting it to X.

leaveoneout	Logical value (passed to density.ppp or fitted.ppm) specifying whether to use a leave-one-out rule when calculating the intensity.
reciplambda	Alternative to lambda. Values of the estimated *reciprocal* $1/\lambda$ of the intensity function. Either a vector giving the reciprocal intensity values at the points of the pattern X, a pixel image (object of class "im") giving the reciprocal intensity values at all locations, or a function(x,y) which can be evaluated to give the reciprocal intensity value at any location.
sigma,varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.
close	Advanced use only. Precomputed data. See section on Advanced Use.

Details

The inhomogeneous pair correlation function $g_{\rm inhom}(r)$ is a summary of the dependence between points in a spatial point process that does not have a uniform density of points.

The best intuitive interpretation is the following: the probability $p(r)$ of finding two points at locations x and y separated by a distance r is equal to

$$p(r) = \lambda(x) lambda(y) g(r) \,{\rm d}x \,{\rm d}y$$

where λ is the intensity function of the point process. For a Poisson point process with intensity function λ, this probability is $p(r) = \lambda(x)\lambda(y)$ so $g_{\rm inhom}(r) = 1$.

The inhomogeneous pair correlation function is related to the inhomogeneous K function through

$$g_{\rm inhom}(r) = \frac{K'_{\rm inhom}(r)}{2\pi r}$$

where $K'_{\rm inhom}(r)$ is the derivative of $K_{\rm inhom}(r)$, the inhomogeneous K function. See Kinhom for information about $K_{\rm inhom}(r)$.

The command pcfinhom estimates the inhomogeneous pair correlation using a modified version of the algorithm in pcf.ppp.

If renormalise=TRUE (the default), then the estimates are multiplied by $c^{\rm normpower}$ where $c = {\rm area}(W)/\sum(1/\lambda(x_i))$. This rescaling reduces the variability and bias of the estimate in small samples and in cases of very strong inhomogeneity. The default value of normpower is 1 but the most sensible value is 2, which would correspond to rescaling the lambda values so that $\sum(1/\lambda(x_i)) = {\rm area}(W)$.

Value

A function value table (object of class "fv"). Essentially a data frame containing the variables

r	the vector of values of the argument r at which the inhomogeneous pair correlation function $g_{\rm inhom}(r)$ has been estimated
theo	vector of values equal to 1, the theoretical value of $g_{\rm inhom}(r)$ for the Poisson process
trans	vector of values of $g_{\rm inhom}(r)$ estimated by translation correction
iso	vector of values of $g_{\rm inhom}(r)$ estimated by Ripley isotropic correction

as required.

Advanced Use

To perform the same computation using several different bandwidths bw, it is efficient to use the argument close. This should be the result of closepairs(X, rmax) for a suitably large value of rmax, namely rmax >= max(r) + 3 * bw.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

pcf, pcf.ppp, bw.stoyan, bw.pcf, Kinhom

Examples

```
data(residualspaper)
X <- residualspaper$Fig4b
plot(pcfinhom(X, stoyan=0.2, sigma=0.1))
fit <- ppm(X, ~polynom(x,y,2))
plot(pcfinhom(X, lambda=fit, normpower=2))
```

---

pcfmulti *Marked pair correlation function*

Description

For a marked point pattern, estimate the multitype pair correlation function using kernel methods.

Usage

```
pcfmulti(X, I, J, ..., r = NULL,
         kernel = "epanechnikov", bw = NULL, stoyan = 0.15,
         correction = c("translate", "Ripley"),
         divisor = c("r", "d"),
         Iname = "points satisfying condition I",
         Jname = "points satisfying condition J")
```

Arguments

X	The observed point pattern, from which an estimate of the cross-type pair correlation function $g_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).
I	Subset index specifying the points of X from which distances are measured.
J	Subset index specifying the points in X to which distances are measured.
...	Ignored.
r	Vector of values for the argument r at which $g(r)$ should be evaluated. There is a sensible default.
kernel	Choice of smoothing kernel, passed to density.default.
bw	Bandwidth for smoothing kernel, passed to density.default.

stoyan	Coefficient for default bandwidth rule.
correction	Choice of edge correction.
divisor	Choice of divisor in the estimation formula: either "r" (the default) or "d".
Iname,Jname	Optional. Character strings describing the members of the subsets I and J.

Details

This is a generalisation of pcfcross to arbitrary collections of points.

The algorithm measures the distance from each data point in subset I to each data point in subset J, excluding identical pairs of points. The distances are kernel-smoothed and renormalised to form a pair correlation function.

- If divisor="r" (the default), then the multitype counterpart of the standard kernel estimator (Stoyan and Stoyan, 1994, pages 284–285) is used. By default, the recommendations of Stoyan and Stoyan (1994) are followed exactly.
- If divisor="d" then a modified estimator is used: the contribution from an interpoint distance d_{ij} to the estimate of $g(r)$ is divided by d_{ij} instead of dividing by r. This usually improves the bias of the estimator when r is close to zero.

There is also a choice of spatial edge corrections (which are needed to avoid bias due to edge effects associated with the boundary of the spatial window): correction="translate" is the Ohser-Stoyan translation correction, and correction="isotropic" or "Ripley" is Ripley's isotropic correction.

The arguments I and J specify two subsets of the point pattern X. They may be any type of subset indices, for example, logical vectors of length equal to npoints(X), or integer vectors with entries in the range 1 to npoints(X), or negative integer vectors.

Alternatively, I and J may be **functions** that will be applied to the point pattern X to obtain index vectors. If I is a function, then evaluating I(X) should yield a valid subset index. This option is useful when generating simulation envelopes using envelope.

The choice of smoothing kernel is controlled by the argument kernel which is passed to density. The default is the Epanechnikov kernel.

The bandwidth of the smoothing kernel can be controlled by the argument bw. Its precise interpretation is explained in the documentation for density.default. For the Epanechnikov kernel with support $[-h, h]$, the argument bw is equivalent to $h/\sqrt{5}$.

If bw is not specified, the default bandwidth is determined by Stoyan's rule of thumb (Stoyan and Stoyan, 1994, page 285) applied to the points of type j. That is, $h = c/\sqrt{\lambda}$, where λ is the (estimated) intensity of the point process of type j, and c is a constant in the range from 0.1 to 0.2. The argument stoyan determines the value of c.

Value

An object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

pcfcross, pcfdot, pcf.ppp.

Examples

```
adult <- (marks(longleaf) >= 30)
juvenile <- !adult
p <- pcfmulti(longleaf, adult, juvenile)
```

`Penttinen` *Penttinen Interaction*

Description

Creates an instance of the Penttinen pairwise interaction point process model, which can then be fitted to point pattern data.

Usage

```
Penttinen(r)
```

Arguments

`r`	circle radius

Details

Penttinen (1984, Example 2.1, page 18), citing Cormack (1979), described the pairwise interaction point process with interaction factor

$$h(d) = e^{\theta A(d)} = \gamma^{A(d)}$$

between each pair of points separated by a distance d. Here $A(d)$ is the area of intersection between two discs of radius r separated by a distance d, normalised so that $A(0) = 1$.

The scale of interaction is controlled by the disc radius r: two points interact if they are closer than $2r$ apart. The strength of interaction is controlled by the canonical parameter θ, which must be less than or equal to zero, or equivalently by the parameter $\gamma = e^{\theta}$, which must lie between 0 and 1.

The potential is inhibitory, i.e.\ this model is only appropriate for regular point patterns. For $\gamma = 0$ the model is a hard core process with hard core diameter $2r$. For $\gamma = 1$ the model is a Poisson process.

The irregular parameter r must be given in the call to `Penttinen`, while the regular parameter θ will be estimated.

This model can be considered as a pairwise approximation to the area-interaction model `AreaInter`.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

References

Cormack, R.M. (1979) Spatial aspects of competition between individuals. Pages 151–212 in *Spatial and Temporal Analysis in Ecology*, eds. R.M. Cormack and J.K. Ord, International Co-operative Publishing House, Fairland, MD, USA.

Penttinen, A. (1984) *Modelling Interaction in Spatial Point Patterns: Parameter Estimation by the Maximum Likelihood Method.* Jyväskylä Studies in Computer Science, Economics and Statistics **7**, University of Jyväskylä, Finland.

See Also

`ppm`, `ppm.object`, `Pairwise`, `AreaInter`.

Examples

```
fit <- ppm(cells ~ 1, Penttinen(0.07))
fit
reach(fit) # interaction range is circle DIAMETER
```

`perimeter` *Perimeter Length of Window*

Description

Computes the perimeter length of a window

Usage

```
perimeter(w)
```

Arguments

`w`	A window (object of class `"owin"`) or data that can be converted to a window by `as.owin`.

Details

This function computes the perimeter (length of the boundary) of the window `w`. If `w` is a rectangle or a polygonal window, the perimeter is the sum of the lengths of the edges of `w`. If `w` is a mask, it is first converted to a polygonal window using `as.polygonal`, then staircase edges are removed using `simplify.owin`, and the perimeter of the resulting polygon is computed.

Value

A numeric value giving the perimeter length of the window.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

area.owin diameter.owin, owin.object, as.owin

Examples

```
perimeter(square(3))
data(letterR)
perimeter(letterR)
if(interactive()) print(perimeter(as.mask(letterR)))
```

---

periodify *Make Periodic Copies of a Spatial Pattern*

Description

Given a spatial pattern (point pattern, line segment pattern, window, etc) make shifted copies of the pattern and optionally combine them to make a periodic pattern.

Usage

```
periodify(X, ...)
## S3 method for class 'ppp'
periodify(X, nx = 1, ny = 1, ...,
          combine=TRUE, warn=TRUE, check=TRUE,
          ix=(-nx):nx, iy=(-ny):ny,
          ixy=expand.grid(ix=ix,iy=iy))
## S3 method for class 'psp'
periodify(X, nx = 1, ny = 1, ...,
          combine=TRUE, warn=TRUE, check=TRUE,
          ix=(-nx):nx, iy=(-ny):ny,
          ixy=expand.grid(ix=ix,iy=iy))
## S3 method for class 'owin'
periodify(X, nx = 1, ny = 1, ...,
          combine=TRUE, warn=TRUE,
          ix=(-nx):nx, iy=(-ny):ny,
          ixy=expand.grid(ix=ix,iy=iy))
```

Arguments

X	An object representing a spatial pattern (point pattern, line segment pattern or window).
nx,ny	Integers. Numbers of additional copies of X in each direction. The result will be a grid of 2 * nx + 1 by 2 * ny + 1 copies of the original object. (Overruled by ix, iy, ixy).
...	Ignored.
combine	Logical flag determining whether the copies should be superimposed to make an object like X (if combine=TRUE) or simply returned as a list of objects (combine=FALSE).
warn	Logical flag determining whether to issue warnings.
check	Logical flag determining whether to check the validity of the combined pattern.

ix, iy
: Integer vectors determining the grid positions of the copies of X. (Overruled by ixy).

ixy
: Matrix or data frame with two columns, giving the grid positions of the copies of X.

Details

Given a spatial pattern (point pattern, line segment pattern, etc) this function makes a number of shifted copies of the pattern and optionally combines them. The function periodify is generic, with methods for various kinds of spatial objects.

The default is to make a 3 by 3 array of copies of X and combine them into a single pattern of the same kind as X. This can be used (for example) to compute toroidal or periodic edge corrections for various operations on X.

If the arguments nx, ny are given and other arguments are missing, the original object will be copied nx times to the right and nx times to the left, then ny times upward and ny times downward, making (2 * nx + 1) * (2 * ny + 1) copies altogether, arranged in a grid, centred on the original object.

If the arguments ix, iy or ixy are specified, then these determine the grid positions of the copies of X that will be made. For example (ix,iy) = (1, 2) means a copy of X shifted by the vector (ix * w, iy * h) where w,h are the width and height of the bounding rectangle of X.

If combine=TRUE (the default) the copies of X are superimposed to create an object of the same kind as X. If combine=FALSE the copies of X are returned as a list.

Value

If combine=TRUE, an object of the same class as X. If combine=FALSE, a list of objects of the same class as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

shift

Examples

```
data(cells)
plot(periodify(cells))
a <- lapply(periodify(Window(cells), combine=FALSE),
      plot, add=TRUE,lty=2)
```

persp.im *Perspective Plot of Pixel Image*

Description

Displays a perspective plot of a pixel image.

Usage

```
  ## S3 method for class 'im'
persp(x, ...,
                      colmap=NULL, colin=x, apron=FALSE, visible=FALSE)
```

Arguments

`x`	The pixel image to be plotted as a surface. An object of class `"im"` (see `im.object`).
`...`	Extra arguments passed to `persp.default` to control the display.
`colmap`	Optional data controlling the colour map. See Details.
`colin`	Optional. Colour input. Another pixel image (of the same dimensions as `x`) containing the values that will be mapped to colours.
`apron`	Logical. If `TRUE`, a grey apron is placed around the sides of the perspective plot.
`visible`	Logical value indicating whether to compute which pixels of `x` are visible in the perspective view. See Details.

Details

This is the `persp` method for the class `"im"`.

The pixel image `x` must have real or integer values. These values are treated as heights of a surface, and the surface is displayed as a perspective plot on the current plot device, using equal scales on the `x` and `y` axes.

The optional argument `colmap` gives an easy way to display different altitudes in different colours (if this is what you want).

- If `colmap` is a colour map (object of class `"colourmap"`, created by the function `colourmap`) then this colour map will be used to associate altitudes with colours.
- If `colmap` is a character vector, then the range of altitudes in the perspective plot will be divided into `length(colmap)` intervals, and those parts of the surface which lie in a particular altitude range will be assigned the corresponding colour from `colmap`.
- If `colmap` is a function in the R language of the form `function(n, ...)`, this function will be called with an appropriate value of `n` to generate a character vector of `n` colours. Examples of such functions are `heat.colors`, `terrain.colors`, `topo.colors` and `cm.colors`.
- If `colmap` is a function in the R language of the form `function(range, ...)` then it will be called with `range` equal to the range of altitudes, to determine the colour values or colour map. Examples of such functions are `beachcolours` and `beachcolourmap`.
- If `colmap` is a list with entries `breaks` and `col`, then `colmap$breaks` determines the breakpoints of the altitude intervals, and `colmap$col` provides the corresponding colours.

Alternatively, if the argument `colin` (*colour input*) is present, then the colour map `colmap` will be applied to the pixel values of `colin` instead of the pixel values of `x`. The result is a perspective view of a surface with heights determined by `x` and colours determined by `colin` (mapped by `colmap`).

If `apron=TRUE`, vertical surface is drawn around the boundary of the perspective plot, so that the terrain appears to have been cut out of a solid material. If colour data were supplied, then the apron is coloured light grey.

Graphical parameters controlling the perspective plot are passed through the `...` arguments directly to the function `persp.default`. See the examples in `persp.default` or in `demo(persp)`.

The vertical scale is controlled by the argument `expand`: setting `expand=1` will interpret the pixel values as being in the same units as the spatial coordinates x and y and represent them at the same scale.

If `visible=TRUE`, the algorithm also computes whether each pixel in `x` is visible in the perspective view. In order to be visible, a pixel must not be obscured by another pixel which lies in front of it (as seen from the viewing direction), and the three-dimensional vector normal to the surface must be pointing toward the viewer. The return value of `persp.im` then has an attribute `"visible"` which is a pixel image, compatible with `x`, with pixel value equal to `TRUE` if the corresponding pixel in `x` is visible, and `FALSE` if it is not visible.

Value

(invisibly) the 3D transformation matrix returned by `persp.default`, together with an attribute `"expand"` which gives the relative scale of the z coordinate.

If argument `visible=TRUE` was given, the return value also has an attribute `"visible"` which is a pixel image, compatible with `x`, with logical values which are *TRUE* when the corresponding pixel is visible in the perspective view, and `FALSE` when it is obscured.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`perspPoints`, `perspLines` for drawing additional points or lines on the surface.

`im.object`, `plot.im`, `contour.im`

Examples

```
# an image
Z <- setcov(owin())
persp(Z, colmap=terrain.colors(128))
co <- colourmap(range=c(0,1), col=rainbow(128))
persp(Z, colmap=co, axes=FALSE, shade=0.3)

## Terrain elevation
persp(bei.extra$elev, colmap=terrain.colors(128),
      apron=TRUE, theta=-30, phi=20,
      zlab="Elevation", main="", ticktype="detailed",
      expand=6)
```

perspPoints	*Draw Points or Lines on a Surface Viewed in Perspective*

Description

After a surface has been plotted in a perspective view using `persp.im`, these functions can be used to draw points or lines on the surface.

Usage

```
perspPoints(x, y=NULL, ..., Z, M)

perspLines(x, y = NULL, ..., Z, M)

perspSegments(x0, y0 = NULL, x1 = NULL, y1 = NULL, ..., Z, M)

perspContour(Z, M, ...,
             nlevels=10, levels=pretty(range(Z), nlevels))
```

Arguments

`x,y`	Spatial coordinates, acceptable to `xy.coords`, for the points or lines on the horizontal plane.
`Z`	Pixel image (object of class `"im"`) specifying the surface heights.
`M`	Projection matrix returned from `persp.im` when `Z` was plotted.
`...`	Graphical arguments passed to `points`, `lines` or `segments` to control the drawing.
`x0,y0,x1,y1`	Spatial coordinates of the line segments, on the horizontal plane. Alternatively `x0` can be a line segment pattern (object of class `"psp"`) and `y0,x1,y1` can be `NULL`.
`nlevels`	Number of contour levels
`levels`	Vector of heights of contours.

Details

After a surface has been plotted in a perspective view, these functions can be used to draw points or lines on the surface.

The user should already have called `persp.im` in the form `M <- persp(Z, visible=TRUE, ...)` to display the perspective view of the surface `Z`.

Only points and lines which are visible from the viewer's standpoint will be drawn.

Value

Same as the return value from `points` or `segments`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

persp.im

Examples

```
M <- persp(bei.extra$elev, colmap=terrain.colors(128),
           apron=TRUE, theta=-30, phi=20,
           zlab="Elevation", main="",
           expand=6, visible=TRUE, shade=0.3)

perspContour(bei.extra$elev, M=M, col="pink", nlevels=12)
perspPoints(bei, Z=bei.extra$elev, M=M, pch=16, cex=0.3, col="chartreuse")
```

---

pixelcentres *Extract Pixel Centres as Point Pattern*

Description

Given a pixel image or binary mask window, extract the centres of all pixels and return them as a point pattern.

Usage

```
pixelcentres(X, W = NULL, ...)
```

Arguments

X	Pixel image (object of class "im") or window (object of class "owin").
W	Optional window to contain the resulting point pattern.
...	Optional arguments defining the pixel resolution.

Details

If the argument X is a pixel image, the result is a point pattern, consisting of the centre of every pixel whose pixel value is not NA.

If X is a window which is a binary mask, the result is a point pattern consisting of the centre of every pixel inside the window (i.e. every pixel for which the mask value is TRUE).

Otherwise, X is first converted to a window, then converted to a mask using as.mask, then handled as above.

Value

A point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

raster.xy

Examples

```
pixelcentres(letterR, dimyx=5)
```

---

pixellate *Convert Spatial Object to Pixel Image*

Description

Convert a spatial object to a pixel image by measuring the amount of stuff in each pixel.

Usage

```
pixellate(x, ...)
```

Arguments

x	Spatial object to be converted. A point pattern (object of class "ppp"), a window (object of class "owin"), a line segment pattern (object of class "psp"), or some other suitable data.
...	Arguments passed to methods.

Details

The function pixellate converts a geometrical object x into a pixel image, by measuring the *amount* of x that is inside each pixel.

If x is a point pattern, pixellate(x) counts the number of points of x falling in each pixel. If x is a window, pixellate(x) measures the area of intersection of each pixel with the window.

The function pixellate is generic, with methods for point patterns (pixellate.ppp), windows (pixellate.owin), and line segment patterns (pixellate.psp), See the separate documentation for these methods.

The related function as.im also converts x into a pixel image, but typically measures only the presence or absence of x inside each pixel.

Value

A pixel image (object of class "im").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

pixellate.ppp, pixellate.owin, pixellate.psp, pixellate.linnet, as.im `pixellate.owin` *Convert Window to Pixel Image*

Description

Convert a window to a pixel image by measuring the area of intersection between the window and each pixel in a raster.

Usage

```
## S3 method for class 'owin'
pixellate(x, W = NULL, ..., DivideByPixelArea=FALSE)
```

Arguments

`x`	Window (object of class `"owin"`) to be converted.
`W`	Optional. Window determining the pixel raster on which the conversion should occur.
`...`	Optional. Extra arguments passed to `as.mask` to determine the pixel raster.
`DivideByPixelArea`	Logical value, indicating whether the resulting pixel values should be divided by the pixel area.

Details

This is a method for the generic function `pixellate`.

It converts a window `x` into a pixel image, by measuring the *amount* of `x` that is inside each pixel.

(The related function `as.im` also converts `x` into a pixel image, but records only the presence or absence of `x` in each pixel.)

The pixel raster for the conversion is determined by the argument `W` and the extra arguments `...`.

- If `W` is given, and it is a binary mask (a window of type `"mask"`) then it determines the pixel raster.
- If `W` is given, but it is not a binary mask (it is a window of another type) then it will be converted to a binary mask using `as.mask(W, ...)`.
- If `W` is not given, it defaults to `as.mask(as.rectangle(x), ...)`

In the second and third cases it would be common to use the argument `dimyx` to control the number of pixels. See the Examples.

The algorithm then computes the area of intersection of each pixel with the window.

The result is a pixel image with pixel entries equal to these intersection areas.

Value

A pixel image (object of class `"im"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

pixellate.ppp, pixellate, as.im

Examples

```
data(letterR)
plot(pixellate(letterR, dimyx=15))
W <- grow.rectangle(as.rectangle(letterR), 0.2)
plot(pixellate(letterR, W, dimyx=15))
```

---

pixellate.ppp *Convert Point Pattern to Pixel Image*

Description

Converts a point pattern to a pixel image. The value in each pixel is the number of points falling in that pixel, and is typically either 0 or 1.

Usage

```
## S3 method for class 'ppp'
pixellate(x, W=NULL, ..., weights = NULL,
                          padzero=FALSE, fractional=FALSE, preserve=FALSE,
                          DivideByPixelArea=FALSE)

## S3 method for class 'ppp'
as.im(X, ...)
```

Arguments

x,X	Point pattern (object of class "ppp").
...	Arguments passed to as.mask to determine the pixel resolution
W	Optional window mask (object of class "owin") determining the pixel raster.
weights	Optional vector of weights associated with the points.
padzero	Logical value indicating whether to set pixel values to zero outside the window.
fractional,preserve	Logical values determining the type of discretisation. See Details.
DivideByPixelArea	Logical value, indicating whether the resulting pixel values should be divided by the pixel area.

Details

The functions pixellate.ppp and as.im.ppp convert a spatial point pattern x into a pixel image, by counting the number of points (or the total weight of points) falling in each pixel.

Calling as.im.ppp is equivalent to calling pixellate.ppp with its default arguments. Note that pixellate.ppp is more general than as.im.ppp (it has additional arguments for greater flexibility).

The functions as.im.ppp and pixellate.ppp are methods for the generic functions as.im and pixellate respectively, for the class of point patterns.

The pixel raster (in which points are counted) is determined by the argument W if it is present (for pixellate.ppp only). In this case W should be a binary mask (a window object of class "owin" with type "mask"). Otherwise the pixel raster is determined by extracting the window containing x and converting it to a binary pixel mask using as.mask. The arguments ... are passed to as.mask to control the pixel resolution.

If weights is NULL, then for each pixel in the mask, the algorithm counts how many points in x fall in the pixel. This count is usually either 0 (for a pixel with no data points in it) or 1 (for a pixel containing one data point) but may be greater than 1. The result is an image with these counts as its pixel values.

If weights is given, it should be a numeric vector of the same length as the number of points in x. For each pixel, the algorithm finds the total weight associated with points in x that fall in the given pixel. The result is an image with these total weights as its pixel values.

By default (if zeropad=FALSE) the resulting pixel image has the same spatial domain as the window of the point pattern x. If zeropad=TRUE then the resulting pixel image has a rectangular domain; pixels outside the original window are assigned the value zero.

The discretisation procedure is controlled by the arguments fractional and preserve.

- The argument fractional specifies how data points are mapped to pixels. If fractional=FALSE (the default), each data point is allocated to the nearest pixel centre. If fractional=TRUE, each data point is allocated with fractional weight to four pixel centres (the corners of a rectangle containing the data point).
- The argument preserve specifies what to do with pixels lying near the boundary of the window, if the window is not a rectangle. If preserve=FALSE (the default), any contributions that are attributed to pixel centres lying outside the window are reset to zero. If preserve=TRUE, any such contributions are shifted to the nearest pixel lying inside the window, so that the total mass is preserved.

Value

A pixel image (object of class "im").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

pixellate, im, as.im, density.ppp, Smooth.ppp.

Examples

```
data(humberside)
plot(pixellate(humberside))
plot(pixellate(humberside, fractional=TRUE))
```

pixellate.psp *Convert Line Segment Pattern to Pixel Image*

Description

Converts a line segment pattern to a pixel image by measuring the length or number of lines intersecting each pixel.

Usage

```
## S3 method for class 'psp'
pixellate(x, W=NULL, ..., weights = NULL,
                         what=c("length", "number"),
                         DivideByPixelArea=FALSE)
```

Arguments

- `x` Line segment pattern (object of class "psp").
- `W` Optional window (object of class "owin") determining the pixel resolution.
- `...` Optional arguments passed to `as.mask` to determine the pixel resolution.
- `weights` Optional vector of weights associated with each line segment.
- `what` String (partially matched) indicating whether to compute the total length of intersection (`what="length"`, the default) or the total number of segments intersecting each pixel (`what="number"`).
- `DivideByPixelArea` Logical value, indicating whether the resulting pixel values should be divided by the pixel area.

Details

This function converts a line segment pattern to a pixel image by computing, for each pixel, the total length of intersection between the pixel and the line segments. Alternatively it can count the number of line segments intersecting each pixel.

This is a method for the generic function `pixellate` for the class of line segment patterns.

The pixel raster is determined by `W` and the optional arguments `...`. If `W` is missing or `NULL`, it defaults to the window containing `x`. Then `W` is converted to a binary pixel mask using `as.mask`. The arguments `...` are passed to `as.mask` to control the pixel resolution.

If `weights` are given, then the length of the intersection between line segment `i` and pixel `j` is multiplied by `weights[i]` before the lengths are summed for each pixel.

Value

A pixel image (object of class "im") with numeric values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

pixellate, as.mask, as.mask.psp.

Use as.mask.psp if you only want to know which pixels are intersected by lines.

Examples

```
X <- psp(runif(10),runif(10), runif(10), runif(10), window=owin())
plot(pixellate(X))
plot(X, add=TRUE)
sum(lengths.psp(X))
sum(pixellate(X))
plot(pixellate(X, what="n"))
```

---

pixelquad *Quadrature Scheme Based on Pixel Grid*

Description

Makes a quadrature scheme with a dummy point at every pixel of a pixel image.

Usage

```
pixelquad(X, W = as.owin(X))
```

Arguments

X	Point pattern (object of class "ppp") containing the data points for the quadrature scheme.
W	Specifies the pixel grid. A pixel image (object of class "im"), a window (object of class "owin"), or anything that can be converted to a window by as.owin.

Details

This is a method for producing a quadrature scheme for use by ppm. It is an alternative to quadscheme.

The function ppm fits a point process model to an observed point pattern using the Berman-Turner quadrature approximation (Berman and Turner, 1992; Baddeley and Turner, 2000) to the pseudo-likelihood of the model. It requires a quadrature scheme consisting of the original data point pattern, an additional pattern of dummy points, and a vector of quadrature weights for all these points. Such quadrature schemes are represented by objects of class "quad". See quad.object for a description of this class.

Given a grid of pixels, this function creates a quadrature scheme in which there is one dummy point at the centre of each pixel. The counting weights are used (the weight attached to each quadrature point is 1 divided by the number of quadrature points falling in the same pixel).

The argument X specifies the locations of the data points for the quadrature scheme. Typically this would be a point pattern dataset.

The argument W specifies the grid of pixels for the dummy points of the quadrature scheme. It should be a pixel image (object of class "im"), a window (object of class "owin"), or anything that can be converted to a window by as.owin. If W is a pixel image or a binary mask (a window of type "mask") then the pixel grid of W will be used. If W is a rectangular or polygonal window, then it will first be converted to a binary mask using as.mask at the default pixel resolution.

Value

An object of class "quad" describing the quadrature scheme (data points, dummy points, and quadrature weights) suitable as the argument Q of the function ppm() for fitting a point process model.

The quadrature scheme can be inspected using the print and plot methods for objects of class "quad".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quadscheme, quad.object, ppm

Examples

```
W <- owin(c(0,1),c(0,1))
X <- runifpoint(42, W)
W <- as.mask(W,dimyx=128)
pixelquad(X,W)
```

---

plot.anylist *Plot a List of Things*

Description

Plots a list of things

Usage

```
## S3 method for class 'anylist'
plot(x, ..., main, arrange=TRUE,
   nrows=NULL, ncols=NULL, main.panel=NULL,
   mar.panel=c(2,1,1,2), hsep=0, vsep=0,
   panel.begin=NULL, panel.end=NULL, panel.args=NULL,
   panel.begin.args=NULL, panel.end.args=NULL, panel.vpad=0.2,
   plotcommand="plot",
   adorn.left=NULL, adorn.right=NULL, adorn.top=NULL, adorn.bottom=NULL,
   adorn.size=0.2, equal.scales=FALSE, halign=FALSE, valign=FALSE)
```

Arguments

x	An object of the class "anylist". Essentially a list of objects.
...	Arguments passed to plot when generating each plot panel.
main	Overall heading for the plot.
arrange	Logical flag indicating whether to plot the objects side-by-side on a single page (arrange=TRUE) or plot them individually in a succession of frames (arrange=FALSE).

`nrows,ncols`	Optional. The number of rows/columns in the plot layout (assuming `arrange=TRUE`). You can specify either or both of these numbers.
`main.panel`	Optional. A character string, or a vector of character strings, giving the headings for each of the objects.
`mar.panel`	Size of the margins outside each plot panel. A numeric vector of length 4 giving the bottom, left, top, and right margins in that order. (Alternatively the vector may have length 1 or 2 and will be replicated to length 4). See the section on *Spacing between plots*.
`hsep,vsep`	Additional horizontal and vertical separation between plot panels, expressed in the same units as `mar.panel`.
`panel.begin,panel.end`	Optional. Functions that will be executed before and after each panel is plotted. See Details.
`panel.args`	Optional. Function that determines different plot arguments for different panels. See Details.
`panel.begin.args`	Optional. List of additional arguments for `panel.begin` when it is a function.
`panel.end.args`	Optional. List of additional arguments for `panel.end` when it is a function.
`panel.vpad`	Amount of extra vertical space that should be allowed for the title of each panel, if a title will be displayed. Expressed as a fraction of the height of the panel. Applies only when `equal.scales=FALSE` (the default) and requires that the height of each panel can be determined.
`plotcommand`	Optional. Character string containing the name of the command that should be executed to plot each panel.
`adorn.left,adorn.right,adorn.top,adorn.bottom`	Optional. Functions (with no arguments) that will be executed to generate additional plots at the margins (left, right, top and/or bottom, respectively) of the array of plots.
`adorn.size`	Relative width (as a fraction of the other panels' widths) of the margin plots.
`equal.scales`	Logical value indicating whether the components should be plotted at (approximately) the same physical scale.
`halign,valign`	Logical values indicating whether panels in a column should be aligned to the same x coordinate system (`halign=TRUE`) and whether panels in a row should be aligned to the same y coordinate system (`valign=TRUE`). These are applicable only if `equal.scales=TRUE`.

Details

This is the `plot` method for the class `"anylist"`.

An object of class `"anylist"` represents a list of objects intended to be treated in the same way. This is the method for `plot`.

In the **spatstat** package, various functions produce an object of class `"anylist"`, essentially a list of objects of the same kind. These objects can be plotted in a nice arrangement using `plot.anylist`. See the Examples.

The argument `panel.args` determines extra graphics parameters for each panel. It should be a function that will be called as `panel.args(i)` where `i` is the panel number. Its return value should be a list of graphics parameters that can be passed to the relevant `plot` method. These parameters override any parameters specified in the `...` arguments.

The arguments `panel.begin` and `panel.end` determine graphics that will be plotted before and after each panel is plotted. They may be objects of some class that can be plotted with the generic `plot` command. Alternatively they may be functions that will be called as `panel.begin(i, y, main=main.panel[i])` and `panel.end(i, y, add=TRUE)` where `i` is the panel number and `y = x[[i]]`.

If all entries of `x` are pixel images, the function `image.listof` is called to control the plotting. The arguments `equal.ribbon` and `col` can be used to determine the colour map or maps applied.

If `equal.scales=FALSE` (the default), then the plot panels will have equal height on the plot device (unless there is only one column of panels, in which case they will have equal width on the plot device). This means that the objects are plotted at different physical scales, by default.

If `equal.scales=TRUE`, then the dimensions of the plot panels on the plot device will be proportional to the spatial dimensions of the corresponding components of `x`. This means that the objects will be plotted at *approximately* equal physical scales. If these objects have very different spatial sizes, the plot command could fail (when it tries to plot the smaller objects at a tiny scale), with an error message that the figure margins are too large.

The objects will be plotted at *exactly* equal physical scales, and *exactly* aligned on the device, under the following conditions:

- every component of `x` is a spatial object whose position can be shifted by `shift`;
- `panel.begin` and `panel.end` are either `NULL` or they are spatial objects whose position can be shifted by `shift`;
- `adorn.left`, `adorn.right`, `adorn.top` and `adorn.bottom` are all `NULL`.

Another special case is when every component of `x` is an object of class `"fv"` representing a function. If `equal.scales=TRUE` then all these functions will be plotted with the same axis scales (i.e. with the same `xlim` and the same `ylim`).

Value

Null.

Spacing between plots

The spacing between individual plots is controlled by the parameters `mar.panel`, `hsep` and `vsep`.

If `equal.scales=FALSE`, the plot panels are logically separate plots. The margins for each panel are determined by the argument `mar.panel` which becomes the graphics parameter `mar` described in the help file for `par`. One unit of `mar` corresponds to one line of text in the margin. If `hsep` or `vsep` are present, `mar.panel` is augmented by `c(vsep, hsep, vsep, hsep)/2`.

If `equal.scales=TRUE`, all the plot panels are drawn in the same coordinate system which represents a physical scale. The unit of measurement for `mar.panel[1,3]` is one-sixth of the greatest height of any object plotted in the same row of panels, and the unit for `mar.panel[2,4]` is one-sixth of the greatest width of any object plotted in the same column of panels. If `hsep` or `vsep` are present, they are interpreted in the same units as `mar.panel[2]` and `mar.panel[1]` respectively.

Error messages

If the error message 'Figure margins too large' occurs, this generally means that one of the objects had a much smaller physical scale than the others. Ensure that `equal.scales=FALSE` and increase the values of `mar.panel`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

contour.listof, image.listof, density.splitppp

Examples

```
 trichotomy <- list(regular=cells,
                    random=japanesepines,
                    clustered=redwood)
 K <- lapply(trichotomy, Kest)
 K <- as.anylist(K)
 plot(K, main="")

# list of 3D point patterns
 ape1 <- osteo[osteo$shortid==4, "pts", drop=TRUE]
 class(ape1)
 plot(ape1, main.panel="", mar.panel=0.1, hsep=0.7, vsep=1,
      cex=1.5, pch=21, bg='white')
```

---

plot.bermantest *Plot Result of Berman Test*

Description

Plot the result of Berman's test of goodness-of-fit

Usage

```
## S3 method for class 'bermantest'
plot(x, ...,
                    lwd=par("lwd"), col=par("col"), lty=par("lty"),
                    lwd0=lwd, col0=2, lty0=2)
```

Arguments

x	Object to be plotted. An object of class "bermantest" produced by berman.test.
...	extra arguments that will be passed to the plotting function plot.ecdf.
col,lwd,lty	The width, colour and type of lines used to plot the empirical distribution curve.
col0,lwd0,lty0	The width, colour and type of lines used to plot the predicted (null) distribution curve.

Details

This is the plot method for the class "bermantest". An object of this class represents the outcome of Berman's test of goodness-of-fit of a spatial Poisson point process model, computed by berman.test.

For the *Z1* test (i.e. if x was computed using berman.test(,which="Z1")), the plot displays the two cumulative distribution functions that are compared by the test: namely the empirical cumulative distribution function of the covariate at the data points, $\hat{F}$, and the predicted cumulative distribution function of the covariate under the model, F_0, both plotted against the value of the covariate. Two vertical lines show the mean values of these two distributions. If the model is correct, the two curves should be close; the test is based on comparing the two vertical lines.

For the *Z2* test (i.e. if x was computed using berman.test(,which="Z2")), the plot displays the empirical cumulative distribution function of the values $U_i = F_0(Y_i)$ where Y_i is the value of the covariate at the i-th data point. The diagonal line with equation $y = x$ is also shown. Two vertical lines show the mean of the values U_i and the value $1/2$. If the model is correct, the two curves should be close. The test is based on comparing the two vertical lines.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

berman.test

Examples

```
# synthetic data: nonuniform Poisson process
X <- rpoispp(function(x,y) { 100 * exp(-x) }, win=square(1))

# fit uniform Poisson process
fit0 <- ppm(X, ~1)

# test covariate = x coordinate
xcoord <- function(x,y) { x }

# test wrong model
k <- berman.test(fit0, xcoord, "Z1")

# plot result of test
plot(k, col="red", col0="green")

# Z2 test
k2 <- berman.test(fit0, xcoord, "Z2")
plot(k2, col="red", col0="green")
```

plot.cdftest *Plot a Spatial Distribution Test*

Description

Plot the result of a spatial distribution test computed by `cdf.test`.

Usage

```
## S3 method for class 'cdftest'
plot(x, ...,
                    style=c("cdf", "PP", "QQ"),
                    lwd=par("lwd"), col=par("col"), lty=par("lty"),
                    lwd0=lwd, col0=2, lty0=2,
                    do.legend)
```

Arguments

`x`	Object to be plotted. An object of class `"cdftest"` produced by a method for `cdf.test`.
`...`	extra arguments that will be passed to the plotting function `plot.default`.
`style`	Style of plot. See Details.
`col,lwd,lty`	The width, colour and type of lines used to plot the empirical curve (the empirical distribution, or PP plot or QQ plot).
`col0,lwd0,lty0`	The width, colour and type of lines used to plot the reference curve (the predicted distribution, or the diagonal).
`do.legend`	Logical value indicating whether to add an explanatory legend. Applies only when `style="cdf"`.

Details

This is the `plot` method for the class `"cdftest"`. An object of this class represents the outcome of a spatial distribution test, computed by `cdf.test`, and based on either the Kolmogorov-Smirnov, Cramér-von Mises or Anderson-Darling test.

If `style="cdf"` (the default), the plot displays the two cumulative distribution functions that are compared by the test: namely the empirical cumulative distribution function of the covariate at the data points, and the predicted cumulative distribution function of the covariate under the model, both plotted against the value of the covariate. The Kolmogorov-Smirnov test statistic (for example) is the maximum vertical separation between the two curves.

If `style="PP"` then the P-P plot is drawn. The x coordinates of the plot are cumulative probabilities for the covariate under the model. The y coordinates are cumulative probabilities for the covariate at the data points. The diagonal line $y = x$ is also drawn for reference. The Kolmogorov-Smirnov test statistic is the maximum vertical separation between the P-P plot and the diagonal reference line.

If `style="QQ"` then the Q-Q plot is drawn. The x coordinates of the plot are quantiles of the covariate under the model. The y coordinates are quantiles of the covariate at the data points. The diagonal line $y = x$ is also drawn for reference. The Kolmogorov-Smirnov test statistic cannot be read off the Q-Q plot.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

cdf.test

Examples

```
op <- options(useFancyQuotes=FALSE)

# synthetic data: nonuniform Poisson process
X <- rpoispp(function(x,y) { 100 * exp(x) }, win=square(1))

# fit uniform Poisson process
fit0 <- ppm(X, ~1)

# test covariate = x coordinate
xcoord <- function(x,y) { x }

# test wrong model
k <- cdf.test(fit0, xcoord)

# plot result of test
plot(k, lwd0=3)

plot(k, style="PP")

plot(k, style="QQ")

options(op)
```

---

plot.colourmap *Plot a Colour Map*

Description

Displays a colour map as a colour ribbon

Usage

```
## S3 method for class 'colourmap'
plot(x, ...,
       main, xlim = NULL, ylim = NULL, vertical = FALSE, axis = TRUE,
       labelmap=NULL, gap=0.25, add=FALSE)
```

Arguments

`x`	Colour map to be plotted. An object of class `"colourmap"`.
`...`	Graphical arguments passed to `image.default` or `axis`.
`main`	Main title for plot. A character string.
`xlim`	Optional range of `x` values for the location of the colour ribbon.
`ylim`	Optional range of `y` values for the location of the colour ribbon.
`vertical`	Logical flag determining whether the colour ribbon is plotted as a horizontal strip (`FALSE`) or a vertical strip (`TRUE`).
`axis`	Logical flag determining whether an axis should be plotted showing the numerical values that are mapped to the colours.
`labelmap`	Function. If this is present, then the labels on the plot, which indicate the input values corresponding to particular colours, will be transformed by `labelmap` before being displayed on the plot. Typically used to simplify or shorten the labels on the plot.
`gap`	Distance between separate blocks of colour, as a fraction of the width of one block, if the colourmap is discrete.
`add`	Logical value indicating whether to add the colourmap to the existing plot (`add=TRUE`), or to start a new plot (`add=FALSE`, the default).

Details

This is the plot method for the class `"colourmap"`. An object of this class (created by the function `colourmap`) represents a colour map or colour lookup table associating colours with each data value.

The command `plot.colourmap` displays the colour map as a colour ribbon or as a colour legend (a sequence of blocks of colour). This plot can be useful on its own to inspect the colour map.

If the domain of the colourmap is an interval of real numbers, the colourmap is displayed as a continuous ribbon of colour. If the domain of the colourmap is a finite set of inputs, the colours are displayed as separate blocks of colour. The separation between blocks is equal to `gap` times the width of one block.

To annotate an existing plot with an explanatory colour ribbon or colour legend, specify `add=TRUE` and use the arguments `xlim` and/or `ylim` to control the physical position of the ribbon on the plot.

Labels explaining the colour map are drawn by `axis` and can be modified by specifying arguments that will be passed to this function.

Value

None.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`colourmap`

Examples

```
co <- colourmap(rainbow(100), breaks=seq(-1,1,length=101))
plot(co)
plot(co, col.ticks="pink")
ca <- colourmap(rainbow(8), inputs=letters[1:8])
plot(ca, vertical=TRUE)
```

---

plot.dppm *Plot a fitted determinantal point process*

Description

Plots a fitted determinantal point process model, displaying the fitted intensity and the fitted summary function.

Usage

```
  ## S3 method for class 'dppm'
plot(x, ..., what=c("intensity", "statistic"))
```

Arguments

x	Fitted determinantal point process model. An object of class "dppm".
...	Arguments passed to plot.ppm and plot.fv to control the plot.
what	Character vector determining what will be plotted.

Details

This is a method for the generic function plot for the class "dppm" of fitted determinantal point process models.

The argument x should be a determinantal point process model (object of class "dppm") obtained using the function dppm.

The choice of plots (and the order in which they are displayed) is controlled by the argument what. The options (partially matched) are "intensity" and "statistic".

This command is capable of producing two different plots:

what="intensity" specifies the fitted intensity of the model, which is plotted using plot.ppm. By default this plot is not produced for stationary models.

what="statistic" specifies the empirical and fitted summary statistics, which are plotted using plot.fv. This is only meaningful if the model has been fitted using the Method of Minimum Contrast, and it is turned off otherwise.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

dppm, plot.ppm,

Examples

```
fit <- dppm(swedishpines ~ x + y, dppGauss())
plot(fit)
```

---

plot.envelope *Plot a Simulation Envelope*

Description

Plot method for the class "envelope".

Usage

```
## S3 method for class 'envelope'
plot(x, ..., main)
```

Arguments

x	An object of class "envelope", containing the variables to be plotted or variables from which the plotting coordinates can be computed.
main	Main title for plot.
...	Extra arguments passed to plot.fv.

Details

This is the plot method for the class "envelope" of simulation envelopes. Objects of this class are created by the command envelope.

This plot method is currently identical to plot.fv.

Its default behaviour is to shade the region between the upper and lower envelopes in a light grey colour. To suppress the shading and plot the upper and lower envelopes as curves, set shade=NULL. To change the colour of the shading, use the argument shadecol which is passed to plot.fv.

See plot.fv for further information on how to control the plot.

Value

Either NULL, or a data frame giving the meaning of the different line types and colours.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

envelope, plot.fv

Examples

```
data(cells)
E <- envelope(cells, Kest, nsim=19)
plot(E)
plot(E, sqrt(./pi) ~ r)
```

---

plot.fasp *Plot a Function Array*

Description

Plots an array of summary functions, usually associated with a point pattern, stored in an object of class "fasp". A method for plot.

Usage

```
   ## S3 method for class 'fasp'
plot(x,formule=NULL, ...,
                           subset=NULL, title=NULL, banner=TRUE,
                           transpose=FALSE,
                           samex=FALSE, samey=FALSE,
                           mar.panel=NULL,
                           outerlabels=TRUE, cex.outerlabels=1.25,
                           legend=FALSE)
```

Arguments

x	An object of class "fasp" representing a function array.
formule	A formula or list of formulae indicating what variables are to be plotted against what variable. Each formula is either an R language formula object, or a string that can be parsed as a formula. If formule is a list, its k^{th} component should be applicable to the $(i,j)^{th}$ plot where x$which[i,j]=k. If the formula is left as NULL, then plot.fasp attempts to use the component default.formula of x. If that component is NULL as well, it gives up.
...	Arguments passed to plot.fv to control the individual plot panels.
subset	A logical vector, or a vector of indices, or an expression or a character string, or a **list** of such, indicating a subset of the data to be included in each plot. If subset is a list, its k^{th} component should be applicable to the $(i,j)^{th}$ plot where x$which[i,j]=k.
title	Overall title for the plot.
banner	Logical. If TRUE, the overall title is plotted. If FALSE, the overall title is not plotted and no space is allocated for it.
transpose	Logical. If TRUE, rows and columns will be exchanged.
samex,samey	Logical values indicating whether all individual plot panels should have the same x axis limits and the same y axis limits, respectively. This makes it easier to compare the plots.
mar.panel	Vector of length 4 giving the value of the graphics parameter mar controlling the size of plot margins for each individual plot panel. See par.

`outerlabels`	Logical. If `TRUE`, the row and column names of the array of functions are plotted in the margins of the array of plot panels. If `FALSE`, each individual plot panel is labelled by its row and column name.
`cex.outerlabels`	Character expansion factor for row and column labels of array.
`legend`	Logical flag determining whether to plot a legend in each panel.

Details

An object of class `"fasp"` represents an array of summary functions, usually associated with a point pattern. See `fasp.object` for details. Such an object is created, for example, by `alltypes`.

The function `plot.fasp` is a method for `plot`. It calls `plot.fv` to plot the individual panels.

For information about the interpretation of the arguments `formule` and `subset`, see `plot.fv`.

Arguments that are often passed through `...` include `col` to control the colours of the different lines in a panel, and `lty` and `lwd` to control the line type and line width of the different lines in a panel. The argument `shade` can also be used to display confidence intervals or significance bands as filled grey shading. See `plot.fv`.

The argument `title`, if present, will determine the overall title of the plot. If it is absent, it defaults to `x$title`. Titles for the individual plot panels will be taken from `x$titles`.

Value

None.

Warnings

(Each component of) the `subset` argument may be a logical vector (of the same length as the vectors of data which are extracted from `x`), or a vector of indices, or an **expression** such as `expression(r<=0.2)`, or a text string, such as `"r<=0.2"`.

Attempting a syntax such as `subset = r<=0.2` (without wrapping `r<=0.2` either in quote marks or in `expression()`) will cause this function to fall over.

Variables referred to in any formula must exist in the data frames stored in `x`. What the names of these variables are will of course depend upon the nature of `x`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`alltypes`, `plot.fv`, `fasp.object`

Examples

```
## Not run:
# Bramble Canes data.
data(bramblecanes)

X.G <- alltypes(bramblecanes,"G",dataname="Bramblecanes",verb=TRUE)
plot(X.G)
plot(X.G,subset="r<=0.2")
```

```
    plot(X.G,formule=asin(sqrt(cbind(km,theo))) ~ asin(sqrt(theo)))
    plot(X.G,fo=cbind(km,theo) - theo~r,subset="r<=0.2")

    # Simulated data.
    pp <- runifpoint(350, owin(c(0,1),c(0,1)))
    pp <- pp %mark% factor(c(rep(1,50),rep(2,100),rep(3,200)))
    X.K <- alltypes(pp,"K",verb=TRUE,dataname="Fake Data")
    plot(X.K,fo=cbind(border,theo)~theo,subset="theo<=0.75")

## End(Not run)
```

---

plot.fv *Plot Function Values*

Description

Plot method for the class "fv".

Usage

```
 ## S3 method for class 'fv'
plot(x, fmla, ..., subset=NULL, lty=NULL, col=NULL, lwd=NULL,
          xlim=NULL, ylim=NULL, xlab=NULL, ylab=NULL, ylim.covers=NULL,
          legend=!add, legendpos="topleft", legendavoid=missing(legendpos),
          legendmath=TRUE, legendargs=list(),
          shade=fvnames(x, ".s"), shadecol="grey",
          add=FALSE, log="",
          mathfont=c("italic", "plain", "bold", "bolditalic"),
          limitsonly=FALSE)
```

Arguments

x	An object of class "fv", containing the variables to be plotted or variables from which the plotting coordinates can be computed.
fmla	an R language formula determining which variables or expressions are plotted. Either a formula object, or a string that can be parsed as a formula. See Details.
subset	(optional) subset of rows of the data frame that will be plotted.
lty	(optional) numeric vector of values of the graphical parameter lty controlling the line style of each plot.
col	(optional) numeric vector of values of the graphical parameter col controlling the colour of each plot.
lwd	(optional) numeric vector of values of the graphical parameter lwd controlling the line width of each plot.
xlim	(optional) range of x axis
ylim	(optional) range of y axis
xlab	(optional) label for x axis
ylab	(optional) label for y axis
...	Extra arguments passed to plot.default.

ylim.covers
: Optional vector of y values that must be included in the y axis. For example `ylim.covers=0` will ensure that the y axis includes the origin.

legend
: Logical flag or `NULL`. If `legend=TRUE`, the algorithm plots a legend in the top left corner of the plot, explaining the meaning of the different line types and colours.

legendpos
: The position of the legend. Either a character string keyword (see `legend` for keyword options) or a pair of coordinates in the format `list(x,y)`. Alternatively if `legendpos="float"`, a location will be selected inside the plot region, avoiding the graphics.

legendavoid
: Whether to avoid collisions between the legend and the graphics. Logical value. If `TRUE`, the code will check for collisions between the legend box and the graphics, and will override `legendpos` if a collision occurs. If `FALSE`, the value of `legendpos` is always respected.

legendmath
: Logical. If `TRUE`, the legend will display the mathematical notation for each curve. If `FALSE`, the legend text is the identifier (column name) for each curve.

legendargs
: Named list containing additional arguments to be passed to `legend` controlling the appearance of the legend.

shade
: A character vector giving the names of two columns of `x`, or another type of index that identifies two columns. When the corresponding curves are plotted, the region between the curves will be shaded in light grey. The object `x` may or may not contain two columns which are designated as boundaries for shading; they are identified by `fvnames(x, ".s")`. The default is to shade between these two curves if they exist. To suppress this behaviour, set `shade=NULL`.

shadecol
: The colour to be used in the `shade` plot. A character string or an integer specifying a colour.

add
: Logical. Whether the plot should be added to an existing plot

log
: A character string which contains `"x"` if the x axis is to be logarithmic, `"y"` if the y axis is to be logarithmic and `"xy"` or `"yx"` if both axes are to be logarithmic.

mathfont
: Character string. The font to be used for mathematical expressions in the axis labels and the legend.

limitsonly
: Logical. If `FALSE`, plotting is performed normally. If `TRUE`, no plotting is performed at all; just the x and y limits of the plot are computed and returned.

Details

This is the `plot` method for the class `"fv"`.

The use of the argument `fmla` is like `plot.formula`, but offers some extra functionality.

The left and right hand sides of `fmla` are evaluated, and the results are plotted against each other (the left side on the y axis against the right side on the x axis).

The left and right hand sides of `fmla` may be the names of columns of the data frame `x`, or expressions involving these names. If a variable in `fmla` is not the name of a column of `x`, the algorithm will search for an object of this name in the environment where `plot.fv` was called, and then in the enclosing environment, and so on.

Multiple curves may be specified by a single formula of the form `cbind(y1,y2,...,yn) ~ x`, where `x,y1,y2,...,yn` are expressions involving the variables in the data frame. Each of the variables `y1,y2,...,yn` in turn will be plotted against `x`. See the examples.

Convenient abbreviations which can be used in the formula are

- the symbol `.` which represents all the columns in the data frame that will be plotted by default;

- the symbol .x which represents the function argument;
- the symbol .y which represents the recommended value of the function.

For further information, see fvnames.

The value returned by this plot function indicates the meaning of the line types and colours in the plot. It can be used to make a suitable legend for the plot if you want to do this by hand. See the examples.

The argument shade can be used to display critical bands or confidence intervals. If it is not NULL, then it should be a subset index for the columns of x, that identifies exactly 2 columns. When the corresponding curves are plotted, the region between the curves will be shaded in light grey. See the Examples.

The default values of lty, col and lwd can be changed using spatstat.options("plot.fv").

Use type = "n" to create the plot region and draw the axes without plotting any data.

Use limitsonly=TRUE to suppress all plotting and just compute the x and y limits. This can be used to calculate common x and y scales for several plots.

To change the kind of parenthesis enclosing the explanatory text about the unit of length, use spatstat.options('units.paren')

Value

Invisible: either NULL, or a data frame giving the meaning of the different line types and colours.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

fv.object, Kest

Examples

```
K <- Kest(cells)
# K is an object of class "fv"

plot(K, iso ~ r)                # plots iso against r

plot(K, sqrt(iso/pi) ~ r)   # plots sqrt(iso/r)  against r

plot(K, cbind(iso,theo) ~ r)   # plots iso against r  AND theo against r

plot(K, .  ~ r)            # plots all available estimates of K against r

plot(K, sqrt(./pi) ~ r)   # plots all estimates of L-function
                          # L(r) = sqrt(K(r)/pi)

plot(K, cbind(iso,theo) ~ r, col=c(2,3))
                                # plots iso against r  in colour 2
                                # and theo against r in colour 3

plot(K, iso ~ r, subset=quote(r < 0.2))
                                # plots iso against r for r < 10
```

```
# Can't remember the names of the columns? No problem..
plot(K, sqrt(./pi) ~ .x)

# making a legend by hand
v <- plot(K, . ~ r, legend=FALSE)
legend("topleft", legend=v$meaning, lty=v$lty, col=v$col)

# significance bands
KE <- envelope(cells, Kest, nsim=19)
plot(KE, shade=c("hi", "lo"))

# how to display two functions on a common scale
Kr <- Kest(redwood)
a <- plot(K, limitsonly=TRUE)
b <- plot(Kr, limitsonly=TRUE)
xlim <- range(a$xlim, b$xlim)
ylim <- range(a$ylim, b$ylim)
opa <- par(mfrow=c(1,2))
plot(K, xlim=xlim, ylim=ylim)
plot(Kr, xlim=xlim, ylim=ylim)
par(opa)
```

---

plot.hyperframe *Plot Entries in a Hyperframe*

Description

Plots the entries in a hyperframe, in a series of panels, one panel for each row of the hyperframe.

Usage

```
   ## S3 method for class 'hyperframe'
plot(x, e, ..., main, arrange=TRUE,
                               nrows=NULL, ncols=NULL,
                               parargs=list(mar=mar * marsize),
                               marsize=1, mar=c(1,1,3,1))
```

Arguments

`x`	Data to be plotted. A hyperframe (object of class "hyperframe", see hyperframe).
`e`	How to plot each row. Optional. An R language call or expression (typically enclosed in quote() that will be evaluated in each row of the hyperframe to generate the plots.
`...`	Extra arguments controlling the plot (when e is missing).
`main`	Overall title for the array of plots.
`arrange`	Logical flag indicating whether to plot the objects side-by-side on a single page (arrange=TRUE) or plot them individually in a succession of frames (arrange=FALSE).
`nrows,ncols`	Optional. The number of rows/columns in the plot layout (assuming arrange=TRUE). You can specify either or both of these numbers.

parargs — Optional list of arguments passed to par before plotting each panel. Can be used to control margin sizes, etc.

marsize — Optional scale parameter controlling the sizes of margins around the panels. Incompatible with parargs.

mar — Optional numeric vector of length 1, 2 or 4 controlling the relative sizes of margins between the panels. Incompatible with parargs.

Details

This is the plot method for the class "hyperframe".

The argument x must be a hyperframe (like a data frame, except that the entries can be objects of any class; see hyperframe).

This function generates a series of plots, one plot for each row of the hyperframe. If arrange=TRUE (the default), then these plots are arranged in a neat array of panels within a single plot frame. If arrange=FALSE, the plots are simply executed one after another.

Exactly what is plotted, and how it is plotted, depends on the argument e. The default (if e is missing) is to plot only the first column of x. Each entry in the first column is plotted using the generic plot command, together with any extra arguments given in ....

If e is present, it should be an R language expression involving the column names of x. (It is typically created using quote or expression.) The expression will be evaluated once for each row of x. It will be evaluated in an environment where each column name of x is interpreted as meaning the object in that column in the current row. See the Examples.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

hyperframe, with.hyperframe

Examples

```
H <- hyperframe(id=1:10)
H$X <- with(H, rpoispp(100))
H$D <- with(H, distmap(X))
# points only
plot(H[,"X"])
plot(H, quote(plot(X, main=id)))
# points superimposed on images
plot(H, quote({plot(D, main=id); plot(X, add=TRUE)}))
```

plot.im *Plot a Pixel Image*

Description

Plot a pixel image.

Usage

```
## S3 method for class 'im'
plot(x, ...,
                   main,
                   add=FALSE, clipwin=NULL,
                   col=NULL, valuesAreColours=NULL, log=FALSE,
                   ncolours=256, gamma=1,
                   ribbon=show.all, show.all=!add,
                   ribside=c("right", "left", "bottom", "top"),
                   ribsep=0.15, ribwid=0.05, ribn=1024,
                   ribscale=1, ribargs=list(), colargs=list(),
                   useRaster=NULL, workaround=FALSE,
                   do.plot=TRUE)

## S3 method for class 'im'
image(x, ...,
                   main,
                   add=FALSE, clipwin=NULL,
                   col=NULL, valuesAreColours=NULL, log=FALSE,
                   ncolours=256, gamma=1,
                   ribbon=show.all, show.all=!add,
                   ribside=c("right", "left", "bottom", "top"),
                   ribsep=0.15, ribwid=0.05, ribn=1024,
                   ribscale=1, ribargs=list(), colargs=list(),
                   useRaster=NULL, workaround=FALSE,
                   do.plot=TRUE)
```

Arguments

`x`	The pixel image to be plotted. An object of class "im" (see `im.object`).
`...`	Extra arguments passed to `image.default` to control the plot. See Details.
`main`	Main title for the plot.
`add`	Logical value indicating whether to superimpose the image on the existing plot (`add=TRUE`) or to initialise a new plot (`add=FALSE`, the default).
`clipwin`	Optional. A window (object of class "owin"). Only this subset of the image will be displayed.
`col`	Colours for displaying the pixel values. Either a character vector of colour values, an object of class `colourmap`, or a `function` as described under Details.
`valuesAreColours`	Logical value. If `TRUE`, the pixel values of `x` are to be interpreted as colour values.

log	Logical value. If TRUE, the colour map will be evenly-spaced on a logarithmic scale.
ncolours	Integer. The default number of colours in the colour map for a real-valued image.
gamma	Exponent for the gamma correction of the colours. A single positive number.
ribbon	Logical flag indicating whether to display a ribbon showing the colour map. Default is TRUE for new plots and FALSE for added plots.
show.all	Logical value indicating whether to display all plot elements including the main title and colour ribbon. Default is TRUE for new plots and FALSE for added plots.
ribside	Character string indicating where to display the ribbon relative to the main image.
ribsep	Factor controlling the space between the ribbon and the image.
ribwid	Factor controlling the width of the ribbon.
ribn	Number of different values to display in the ribbon.
ribscale	Rescaling factor for tick marks. The values on the numerical scale printed beside the ribbon will be multiplied by this rescaling factor.
ribargs	List of additional arguments passed to image.default, axis and axisTicks to control the display of the ribbon and its scale axis. These may override the ... arguments.
colargs	List of additional arguments passed to col if it is a function.
useRaster	Logical value, passed to image.default. Images are plotted using a bitmap raster if useRaster=TRUE or by drawing polygons if useRaster=FALSE. Bitmap raster display tends to produce better results, but is not supported on all graphics devices. The default is to use bitmap raster display if it is supported.
workaround	Logical value, specifying whether to use a workaround to avoid a bug which occurs with some device drivers in R, in which the image has the wrong spatial orientation. See the section on **Image is Displayed in Wrong Spatial Orientation** below.
do.plot	Logical value indicating whether to actually plot the image and colour ribbon. Setting do.plot=FALSE will simply return the colour map and the bounding box that were chosen for the plot.

Details

This is the plot method for the class "im". [It is also the image method for "im".]

The pixel image x is displayed on the current plot device, using equal scales on the x and y axes.

If ribbon=TRUE, a legend will be plotted. The legend consists of a colour ribbon and an axis with tick-marks, showing the correspondence between the pixel values and the colour map.

Arguments ribside, ribsep, ribwid control the placement of the colour ribbon. By default, the ribbon is placed at the right of the main image. This can be changed using the argument ribside. The width of the ribbon is ribwid times the size of the pixel image, where 'size' means the larger of the width and the height. The distance separating the ribbon and the image is ribsep times the size of the pixel image.

The ribbon contains the colours representing ribn different numerical values, evenly spaced between the minimum and maximum pixel values in the image x, rendered according to the chosen colour map.

The argument ribargs controls the annotation of the colour ribbon. It is a list of arguments to be passed to image.default, axis and axisTicks. To plot the colour ribbon without the axis and tick-marks, use `ribargs=list(axes=FALSE)`. To ensure that the numerals or symbols printed next to the colour map are oriented horizontally, use `ribargs=list(las=1)`. To control the number of tick-marks, use `ribargs=list(nint=N)` where `N` is the desired number of intervals (so there will be `N+1` tickmarks, subject to the vagaries of R internal code).

The argument `ribscale` is used to rescale the numerals printed next to the colour map.

Normally the pixel values are displayed using the colours given in the argument `col`. This may be either

- an explicit colour map (an object of class `"colourmap"`, created by the command `colourmap`). This is the best way to ensure that when we plot different images, the colour maps are consistent.
- a character vector or integer vector that specifies a set of colours. The colour mapping will be stretched to match the range of pixel values in the image `x`. The mapping of pixel values to colours is determined as follows.

  logical-valued images: the values `FALSE` and `TRUE` are mapped to the colours `col[1]` and `col[2]` respectively. The vector `col` should have length 2.

  factor-valued images: the factor levels `levels(x)` are mapped to the entries of `col` in order. The vector `col` should have the same length as `levels(x)`.

  numeric-valued images: By default, the range of pixel values in `x` is divided into `n = length(col)` equal subintervals, which are mapped to the colours in `col`. (If `col` was not specified, it defaults to a vector of 255 colours.)
  Alternatively if the argument `zlim` is given, it should be a vector of length 2 specifying an interval of real numbers. This interval will be used instead of the range of pixel values. The interval from `zlim[1]` to `zlim[2]` will be mapped to the colours in `col`. This facility enables the user to plot several images using a consistent colour map.
  Alternatively if the argument `breaks` is given, then this specifies the endpoints of the subintervals that are mapped to each colour. This is incompatible with `zlim`.
  The arguments `col` and `zlim` or `breaks` are then passed to the function `image.default`.
  For examples of the use of these arguments, see `image.default`.

- a `function` in the R language with an argument named `range` or `inputs`.

  If `col` is a function with an argument named `range`, and if the pixel values of `x` are numeric values, then the colour values will be determined by evaluating `col(range=range(x))`. The result of this evaluation should be a character vector containing colour values, or a `"colourmap"` object. Examples of such functions are `beachcolours` and `beachcolourmap`.

  If `col` is a function with an argument named `inputs`, and if the pixel values of `x` are discrete values (integer, logical, factor or character), then the colour values will be determined by evaluating `col(inputs=p)` where `p` is the set of possible pixel values. The result should be a character vector containing colour values, or a `"colourmap"` object.

- a `function` in the R language with first argument named `n`. The colour values will be determined by evaluating `col(n)` where `n` is the number of distinct pixel values, up to a maximum of 128. The result of this evaluation should be a character vector containing color values. Examples of such functions are `heat.colors`, `terrain.colors`, `topo.colors` and `cm.colors`.

If `spatstat.options("monochrome")` has been set to `TRUE` then **all colours will be converted to grey scale values**.

Other graphical parameters controlling the display of both the pixel image and the ribbon can be passed through the `...` arguments to the function `image.default`. A parameter is handled only if it is one of the following:

- a formal argument of `image.default` that is operative when `add=TRUE`.

- one of the parameters "main", "asp", "sub", "axes", "ann", "cex", "font", "cex.axis", "cex.lab
described in par.
- the argument box, a logical value specifying whether a box should be drawn.

Images are plotted using a bitmap raster if useRaster=TRUE or by drawing polygons if useRaster=FALSE. Bitmap raster display (performed by rasterImage) tends to produce better results, but is not supported on all graphics devices. The default is to use bitmap raster display if it is supported according to dev.capabilities.

Alternatively, the pixel values could be directly interpretable as colour values in R. That is, the pixel values could be character strings that represent colours, or values of a factor whose levels are character strings representing colours.

- If valuesAreColours=TRUE, then the pixel values will be interpreted as colour values and displayed using these colours.
- If valuesAreColours=FALSE, then the pixel values will *not* be interpreted as colour values, even if they could be.
- If valuesAreColours=NULL, the algorithm will guess what it should do. If the argument col is given, the pixel values will *not* be interpreted as colour values. Otherwise, if all the pixel values are strings that represent colours, then they will be interpreted and displayed as colours.

If pixel values are interpreted as colours, the arguments col and ribbon will be ignored, and a ribbon will not be plotted.

Value

The colour map used. An object of class "colourmap".

Also has an attribute "bbox" giving a bounding box for the colour image (including the ribbon if present).

Complex-valued images

If the pixel values in x are complex numbers, they will be converted into four images containing the real and imaginary parts and the modulus and argument, and plotted side-by-side using plot.imlist.

Monochrome colours

If spatstat.options("monochrome") has been set to TRUE, then **the image will be plotted in greyscale**. The colours are converted to grey scale values using to.grey. The choice of colour map still has an effect, since it determines the final grey scale values.

Monochrome display can also be achieved by setting the graphics device parameter colormodel="grey" when starting a new graphics device, or in a call to ps.options or pdf.options.

Image Rendering Errors and Problems

The help for image.default and rasterImage explains that errors may occur, or images may be rendered incorrectly, on some devices, depending on the availability of colours and other device-specific constraints.

If the image is not displayed at all, try setting useRaster=FALSE in the call to plot.im. If the ribbon colours are not displayed, set ribargs=list(useRaster=FALSE).

Errors may occur on some graphics devices if the image is very large. If this happens, try setting useRaster=FALSE in the call to plot.im.

The error message `useRaster=TRUE can only be used with a regular grid` means that the x and y coordinates of the pixels in the image are not perfectly equally spaced, due to numerical rounding. This occurs with some images created by earlier versions of **spatstat**. To repair the coordinates in an image `X`, type `X <- as.im(X)`.

Image is Displayed in Wrong Spatial Orientation

If the image is displayed in the wrong spatial orientation, and you created the image data directly, please check that you understand the **spatstat** convention for the spatial orientation of pixel images. The row index of the matrix of pixel values corresponds to the increasing y coordinate; the column index of the matrix corresponds to the increasing x coordinate (Baddeley, Rubak and Turner, 2015, section 3.6.3, pages 66–67).

Images can be displayed in the wrong spatial orientation on some devices, due to a bug in the device driver. This occurs only when the plot coordinates are *reversed*, that is, when the plot was initialised with coordinate limits `xlim, ylim` such that `xlim[1] > xlim[2]` or `ylim[1] > ylim[2]` or both. This bug is reported to occur only when `useRaster=TRUE`. To fix this, try setting `workaround=TRUE`, or if that is unsuccessful, `useRaster=FALSE`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

See Also

`im.object`, `colourmap`, `contour.im`, `persp.im`, `hist.im`, `image.default`, `spatstat.options`

Examples

```
# an image
Z <- setcov(owin())
plot(Z)
plot(Z, ribside="bottom")
# stretchable colour map
plot(Z, col=terrain.colors(128), axes=FALSE)
# fixed colour map
tc <- colourmap(rainbow(128), breaks=seq(-1,2,length=129))
plot(Z, col=tc)
# colour map function, with argument 'range'
plot(Z, col=beachcolours, colargs=list(sealevel=0.5))
# tweaking the plot
plot(Z, main="La vie en bleu", col.main="blue", cex.main=1.5,
     box=FALSE,
     ribargs=list(col.axis="blue", col.ticks="blue", cex.axis=0.75))
# log scale
V <- eval.im(exp(exp(Z+2))/1e4)
plot(V, log=TRUE, main="Log scale")
# it's complex
Y <- exp(Z + V * 1i)
plot(Y)
```

plot.imlist *Plot a List of Images*

Description

Plots an array of pixel images.

Usage

```
  ## S3 method for class 'imlist'
plot(x, ..., plotcommand="image",
                          equal.ribbon=FALSE, ribmar=NULL)

  ## S3 method for class 'imlist'
image(x, ..., equal.ribbon=FALSE, ribmar=NULL)

  ## S3 method for class 'listof'
image(x, ..., equal.ribbon=FALSE, ribmar=NULL)
```

Arguments

`x`	An object of the class `"imlist"` representing a list of pixel images. Alternatively `x` may belong to the outdated class `"listof"`.
`...`	Arguments passed to `plot.solist` to control the spatial arrangement of panels, and arguments passed to `plot.im` to control the display of each panel.
`equal.ribbon`	Logical. If `TRUE`, the colour maps of all the images will be the same. If `FALSE`, the colour map of each image is adjusted to the range of values of that image.
`ribmar`	Numeric vector of length 4 specifying the margins around the colour ribbon, if `equal.ribbon=TRUE`. Entries in the vector give the margin at the bottom, left, top, and right respectively, as a multiple of the height of a line of text.
`plotcommand`	Character string giving the name of a function to be used to display each image. Recognised by `plot.imlist` only.

Details

These are methods for the generic plot commands `plot` and `image` for the class `"imlist"`. They are currently identical.

An object of class `"imlist"` represents a list of pixel images. (The outdated class `"listof"` is also handled.)

Each entry in the list `x` will be displayed as a pixel image, in an array of panels laid out on the same graphics display, using `plot.solist`. Individual panels are plotted by `plot.im`.

If `equal.ribbon=FALSE` (the default), the images are rendered using different colour maps, which are displayed as colour ribbons beside each image. If `equal.ribbon=TRUE`, the images are rendered using the same colour map, and a single colour ribbon will be displayed at the right side of the array. The colour maps and the placement of the colour ribbons are controlled by arguments `...` passed to `plot.im`.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

plot.solist, plot.im

Examples

```
D <- density(split(amacrine))
image(D, equal.ribbon=TRUE, main="", col.ticks="red", col.axis="red")
```

---

plot.influence.ppm *Plot Influence Measure*

Description

Plots an influence measure that has been computed by influence.ppm.

Usage

```
## S3 method for class 'influence.ppm'
plot(x, ..., multiplot=TRUE)
```

Arguments

x	Influence measure (object of class "influence.ppm") computed by influence.ppm.
...	Arguments passed to plot.ppp to control the plotting.
multiplot	Logical value indicating whether it is permissible to plot more than one panel. This happens if the original point process model is multitype.

Details

This is the plot method for objects of class "influence.ppm". These objects are computed by the command influence.ppm.

For a point process model fitted by maximum likelihood or maximum pseudolikelihood (the default), influence values are associated with the data points. The display shows circles centred at the data points with radii proportional to the influence values. If the original data were a multitype point pattern, then if multiplot=TRUE (the default), there is one such display for each possible type of point, while if multiplot=FALSE there is a single plot combining all data points regardless of type.

For a model fitted by logistic composite likelihood (method="logi" in ppm) influence values are associated with the data points and also with the dummy points used to fit the model. The display consist of two panels, for the data points and dummy points respectively, showing circles with radii proportional to the influence values. If the original data were a multitype point pattern, then if multiplot=TRUE (the default), there is one pair of panels for each possible type of point, while if multiplot=FALSE there is a single plot combining all data and dummy points regardless of type.

Use the argument clipwin to restrict the plot to a subset of the full data.

Value

None.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A. and Chang, Y.M. and Song, Y. (2013) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics* **40**, 86–104.

See Also

influence.ppm

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
plot(influence(fit))
```

---

plot.kppm *Plot a fitted cluster point process*

Description

Plots a fitted cluster point process model, displaying the fitted intensity and the fitted K-function.

Usage

```
## S3 method for class 'kppm'
plot(x, ...,
                     what=c("intensity", "statistic", "cluster"),
                     pause=interactive(),
                     xname)
```

Arguments

x	Fitted cluster point process model. An object of class "kppm".
...	Arguments passed to plot.ppm and plot.fv to control the plot.
what	Character vector determining what will be plotted.
pause	Logical value specifying whether to pause between plots.
xname	Optional. Character string. The name of the object x for use in the title of the plot.

Details

This is a method for the generic function `plot` for the class "kppm" of fitted cluster point process models.

The argument `x` should be a cluster point process model (object of class "kppm") obtained using the function `kppm`.

The choice of plots (and the order in which they are displayed) is controlled by the argument `what`. The options (partially matched) are "intensity", "statistic" and "cluster".

This command is capable of producing three different plots:

what="intensity" specifies the fitted intensity of the model, which is plotted using `plot.ppm`. By default this plot is not produced for stationary models.

what="statistic" specifies the empirical and fitted summary statistics, which are plotted using `plot.fv`. This is only meaningful if the model has been fitted using the Method of Minimum Contrast, and it is turned off otherwise.

what="cluster" specifies a fitted cluster, which is computed by `clusterfield` and plotted by `plot.im`. It is only meaningful for Poisson cluster (incl. Neyman-Scott) processes, and it is turned off for log-Gaussian Cox processes (LGCP). If the model is stationary (and non-LGCP) this option is turned on by default and shows a fitted cluster positioned at the centroid of the observation window. For non-stationary (and non-LGCP) models this option is only invoked if explicitly told so, and in that case an additional argument `locations` (see `clusterfield`) must be given to specify where to position the parent point(s) .

Alternatively `what="all"` selects all available options.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`kppm`, `plot.ppm`,

Examples

```
data(redwood)
fit <- kppm(redwood~1, "Thomas")
plot(fit)
```

`plot.laslett` *Plot Laslett Transform*

Description

Plot the result of Laslett's Transform.

Usage

```
## S3 method for class 'laslett'
plot(x, ...,
          Xpars = list(box = TRUE, col = "grey"),
          pointpars = list(pch = 3, cols = "blue"),
          rectpars = list(lty = 3, border = "green"))
```

Arguments

`x`	Object of class `"laslett"` produced by `laslett` representing the result of Laslett's transform.
`...`	Additional plot arguments passed to `plot.solist`.
`Xpars`	A list of plot arguments passed to `plot.owin` or `plot.im` to display the original region `X` before transformation.
`pointpars`	A list of plot arguments passed to `plot.ppp` to display the tangent points.
`rectpars`	A list of plot arguments passed to `plot.owin` to display the maximal rectangle.

Details

This is the `plot` method for the class `"laslett"`.

The function `laslett` applies Laslett's Transform to a spatial region `X` and returns an object of class `"laslett"` representing the result of the transformation. The result is plotted by this method.

The plot function `plot.solist` is used to align the before-and-after pictures. See `plot.solist` for further options to control the plot.

Value

None.

Author(s)

Kassel Hingee and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

`laslett`

Examples

```
b <- laslett(heather$coarse, plotit=FALSE)
plot(b, main="Heather Data")
```

plot.layered *Layered Plot*

Description

Generates a layered plot. The plot method for objects of class "layered".

Usage

```
## S3 method for class 'layered'
plot(x, ..., which = NULL, plotargs = NULL,
                         add=FALSE, show.all=!add, main=NULL,
                         do.plot=TRUE)
```

Arguments

x	An object of class "layered" created by the function layered.
...	Arguments to be passed to the plot method for *every* layer.
which	Subset index specifying which layers should be plotted.
plotargs	Arguments to be passed to the plot methods for individual layers. A list of lists of arguments of the form name=value.
add	Logical value indicating whether to add the graphics to an existing plot.
show.all	Logical value indicating whether the *first* layer should be displayed in full (including the main title, bounding window, coordinate axes, colour ribbon, and so on).
main	Main title for the plot
do.plot	Logical value indicating whether to actually do the plotting.

Details

Layering is a simple mechanism for controlling a high-level plot that is composed of several successive plots, for example, a background and a foreground plot. The layering mechanism makes it easier to plot, to switch on or off the plotting of each individual layer, to control the plotting arguments that are passed to each layer, and to zoom in on a subregion.

The layers of data to be plotted should first be converted into a single object of class "layered" using the function layered. Then the layers can be plotted using the method plot.layered.

To zoom in on a subregion, apply the subset operator [.layered to x before plotting.

Graphics parameters for each layer are determined by (in order of precedence) ..., plotargs, and layerplotargs(x).

The graphics parameters may also include the special argument .plot specifying (the name of) a function which will be used to perform the plotting instead of the generic plot.

The argument show.all is recognised by many plot methods in **spatstat**. It determines whether a plot is drawn with all its additional components such as the main title, bounding window, coordinate axes, colour ribbons and legends. The default is TRUE for new plots and FALSE for added plots.

In plot.layered, the argument show.all applies only to the **first** layer. The subsequent layers are plotted with show.all=FALSE.

To override this, that is, if you really want to draw all the components of **all** layers of x, insert the argument show.all=TRUE in each entry of plotargs or layerplotargs(x).

Value

(Invisibly) a list containing the return values from the plot commands for each layer. This list has an attribute "bbox" giving a bounding box for the entire plot.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

layered, layerplotargs, [.layered, plot.

Examples

```
data(cells)
D <- distmap(cells)
L <- layered(D, cells)
plot(L)
plot(L, which = 2)
plot(L, plotargs=list(list(ribbon=FALSE), list(pch=3, cols="white")))
# plot a subregion
plot(L[, square(0.5)])
```

---

plot.leverage.ppm *Plot Leverage Function*

Description

Generate a pixel image plot, or a contour plot, or a perspective plot, of a leverage function that has been computed by leverage.ppm.

Usage

```
## S3 method for class 'leverage.ppm'
plot(x, ...,
                              what=c("smooth", "nearest", "exact"),
                              showcut=TRUE,
                              args.cut=list(drawlabels=FALSE),
                              multiplot=TRUE)

## S3 method for class 'leverage.ppm'
contour(x, ...,
                              what=c("smooth", "nearest"),
                              showcut=TRUE,
                              args.cut=list(col=3, lwd=3, drawlabels=FALSE),
                              multiplot=TRUE)

## S3 method for class 'leverage.ppm'
persp(x, ...,
                what=c("smooth", "nearest"),
                main, zlab="leverage")
```

Arguments

x	Leverage function (object of class "leverage.ppm") computed by leverage.ppm.
...	Arguments passed to plot.im or contour.im or persp.im controlling the plot.
what	Character string (partially matched) specifying the values to be plotted. See Details.
showcut	Logical. If TRUE, a contour line is plotted at the level equal to the theoretical mean of the leverage.
args.cut	Optional list of arguments passed to contour.default to control the plotting of the contour line for the mean leverage.
multiplot	Logical value indicating whether it is permissible to display several plot panels.
main	Optional main title. A character string or character vector.
zlab	Label for the z axis. A character string.

Details

These functions are the plot, contour and persp methods for objects of class "leverage.ppm". Such objects are computed by the command leverage.ppm.

The plot method displays the leverage function as a colour pixel image using plot.im, and draws a single contour line at the mean leverage value using contour.default. Use the argument clipwin to restrict the plot to a subset of the full data.

The contour method displays the leverage function as a contour plot, and also draws a single contour line at the mean leverage value, using contour.im.

The persp method displays the leverage function as a surface in perspective view, using persp.im.

Since the exact values of leverage are computed only at a finite set of quadrature locations, there are several options for these plots:

what="smooth": (the default) an image plot showing a smooth function, obtained by applying kernel smoothing to the exact leverage values;

what="nearest": an image plot showing a piecewise-constant function, obtained by taking the exact leverage value at the nearest quadrature point;

what="exact": a symbol plot showing the exact values of leverage as circles, centred at the quadrature points, with diameters proportional to leverage.

The pixel images are already contained in the object x and were computed by leverage.ppm; the resolution of these images is controlled by arguments to leverage.ppm.

Value

Same as for plot.im, contour.im and persp.im respectively.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Chang, Y.M. and Song, Y. (2013) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics* **40**, 86–104.

See Also

leverage.ppm.

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X ~x+y)
lef <- leverage(fit)
plot(lef)
contour(lef)
persp(lef)
```

---

plot.linim *Plot Pixel Image on Linear Network*

Description

Given a pixel image on a linear network, the pixel values are displayed either as colours or as line widths.

Usage

```
## S3 method for class 'linim'
plot(x, ..., style = c("colour", "width"),
             scale, adjust = 1,
             negative.args = list(col=2),
             legend=TRUE,
             leg.side=c("right", "left", "bottom", "top"),
             leg.sep=0.1,
             leg.wid=0.1,
             leg.args=list(),
             leg.scale=1,
             zlim,
             do.plot=TRUE)
```

Arguments

`x`	The pixel image to be plotted. An object of class `"linim"`.
`...`	Extra graphical parameters, passed to `plot.im` if `style="colour"`, or to `polygon` if `style="width"`.
`style`	Character string specifying the type of plot. See Details.
`scale`	Physical scale factor for representing the pixel values as line widths.
`adjust`	Adjustment factor for the default scale.
`negative.args`	A list of arguments to be passed to `polygon` specifying how to plot negative values of `x` when `style="width"`.
`legend`	Logical value indicating whether to plot a legend (colour ribbon or scale bar).
`leg.side`	Character string indicating where to display the legend relative to the main image.
`leg.sep`	Factor controlling the space between the legend and the image.

`leg.wid`	Factor controlling the width of the legend.
`leg.scale`	Rescaling factor for annotations on the legend. The values on the numerical scale printed beside the legend will be multiplied by this rescaling factor.
`leg.args`	List of additional arguments passed to `image.default`, `axis` or `text.default` to control the display of the legend. These may override the `...` arguments.
`zlim`	The range of numerical values that should be mapped. A numeric vector of length 2. Defaults to the range of values of `x`.
`do.plot`	Logical value indicating whether to actually perform the plot.

Details

This is the `plot` method for objects of class `"linim"`. Such an object represents a pixel image defined on a linear network.

If `style="colour"` (the default) then the pixel values of `x` are plotted as colours, using `plot.im`.

If `style="width"` then the pixel values of `x` are used to determine the widths of thick lines centred on the line segments of the linear network.

Value

If `style="colour"`, the result is an object of class `"colourmap"` specifying the colour map used. If `style="width"`, the result is a numeric value `v` giving the physical scale: one unit of pixel value is represented as `v` physical units on the plot.

The result also has an attribute `"bbox"` giving a bounding box for the plot. The bounding box includes the ribbon or scale bar, if present, but not the main title.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

See Also

`linim`, `plot.im`, `polygon`

Examples

```
X <- linfun(function(x,y,seg,tp){y^2+x}, simplenet)
X <- as.linim(X)

plot(X)
plot(X, style="width", main="Width proportional to function value")

# signed values
f <- linfun(function(x,y,seg,tp){y-x}, simplenet)
plot(f, style="w", main="Negative values in red")

plot(f, style="w", negative.args=list(density=10),
     main="Negative values are hatched")
```

plot.linnet *Plot a linear network*

Description

Plots a linear network

Usage

```
 ## S3 method for class 'linnet'
plot(x, ..., main=NULL, add=FALSE,
                              vertices=FALSE, window=FALSE,
                              do.plot=TRUE)
```

Arguments

x	Linear network (object of class "linnet").
...	Arguments passed to plot.psp controlling the plot.
main	Main title for plot. Use main="" to suppress it.
add	Logical. If codeTRUE, superimpose the graphics over the current plot. If FALSE, generate a new plot.
vertices	Logical. Whether to plot the vertices as well.
window	Logical. Whether to plot the window containing the linear network.
do.plot	Logical. Whether to actually perform the plot.

Details

This is the plot method for class "linnet".

Value

An (invisible) object of class "owin" giving the bounding box of the network.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

linnet

Examples

```
plot(simplenet)
```

plot.lintess *Plot a Tessellation on a Linear Network*

Description

Plot a tessellation or division of a linear network into tiles.

Usage

```
## S3 method for class 'lintess'
plot(x, ...,
        main, add = FALSE, style = c("segments", "image"), col = NULL)
```

Arguments

x	Tessellation on a linear network (object of class "lintess").
...	Arguments passed to segments (if style="segments") or to plot.im (if style="image") to control the plot.
main	Optional main title for the plot.
add	Logical value indicating whether the plot is to be added to an existing plot.
style	Character string (partially matched) indicating whether to plot the tiles of the tessellation using segments or to convert the tessellation to a pixel image and use plot.im.
col	Vector of colours, or colour map, determining the colours used to plot the different tiles of the tessellation.

Details

A tessellation on a linear network L is a partition of the network into non-overlapping pieces (tiles). Each tile consists of one or more line segments which are subsets of the line segments making up the network. A tile can consist of several disjoint pieces.

This function plots the tessellation on the current device. It is a method for the generic plot.

If style="segments", each tile is plotted using segments. Colours distinguish the different tiles.

If style="image", the tessellation is converted to a pixel image, and plotted using plot.im.

Value

(Invisible) colour map.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

lintess

Examples

```
X <- runiflpp(7, simplenet)
Z <- divide.linnet(X)
plot(Z, main="tessellation on network")
points(as.ppp(X))
```

---

plot.listof *Plot a List of Things*

Description

Plots a list of things

Usage

```
## S3 method for class 'listof'
plot(x, ..., main, arrange=TRUE,
   nrows=NULL, ncols=NULL, main.panel=NULL,
   mar.panel=c(2,1,1,2), hsep=0, vsep=0,
   panel.begin=NULL, panel.end=NULL, panel.args=NULL,
   panel.begin.args=NULL, panel.end.args=NULL, panel.vpad=0.2,
   plotcommand="plot",
   adorn.left=NULL, adorn.right=NULL, adorn.top=NULL, adorn.bottom=NULL,
   adorn.size=0.2, equal.scales=FALSE, halign=FALSE, valign=FALSE)
```

Arguments

`x`	An object of the class `"listof"`. Essentially a list of objects.
`...`	Arguments passed to `plot` when generating each plot panel.
`main`	Overall heading for the plot.
`arrange`	Logical flag indicating whether to plot the objects side-by-side on a single page (`arrange=TRUE`) or plot them individually in a succession of frames (`arrange=FALSE`).
`nrows,ncols`	Optional. The number of rows/columns in the plot layout (assuming `arrange=TRUE`). You can specify either or both of these numbers.
`main.panel`	Optional. A character string, or a vector of character strings, giving the headings for each of the objects.
`mar.panel`	Size of the margins outside each plot panel. A numeric vector of length 4 giving the bottom, left, top, and right margins in that order. (Alternatively the vector may have length 1 or 2 and will be replicated to length 4). See the section on *Spacing between plots*.
`hsep,vsep`	Additional horizontal and vertical separation between plot panels, expressed in the same units as `mar.panel`.
`panel.begin,panel.end`	Optional. Functions that will be executed before and after each panel is plotted. See Details.
`panel.args`	Optional. Function that determines different plot arguments for different panels. See Details.

panel.begin.args
: Optional. List of additional arguments for `panel.begin` when it is a function.

panel.end.args
: Optional. List of additional arguments for `panel.end` when it is a function.

panel.vpad
: Amount of extra vertical space that should be allowed for the title of each panel, if a title will be displayed. Expressed as a fraction of the height of the panel. Applies only when `equal.scales=FALSE` (the default) and requires that the height of each panel can be determined.

plotcommand
: Optional. Character string containing the name of the command that should be executed to plot each panel.

adorn.left,adorn.right,adorn.top,adorn.bottom
: Optional. Functions (with no arguments) that will be executed to generate additional plots at the margins (left, right, top and/or bottom, respectively) of the array of plots.

adorn.size
: Relative width (as a fraction of the other panels' widths) of the margin plots.

equal.scales
: Logical value indicating whether the components should be plotted at (approximately) the same physical scale.

halign,valign
: Logical values indicating whether panels in a column should be aligned to the same x coordinate system (`halign=TRUE`) and whether panels in a row should be aligned to the same y coordinate system (`valign=TRUE`). These are applicable only if `equal.scales=TRUE`.

Details

This is the `plot` method for the class `"listof"`.

An object of class `"listof"` (defined in the base R package) represents a list of objects, all belonging to a common class. The base R package defines a method for printing these objects, `print.listof`, but does not define a method for `plot`. So here we have provided a method for `plot`.

In the **spatstat** package, various functions produce an object of class `"listof"`, essentially a list of spatial objects of the same kind. These objects can be plotted in a nice arrangement using `plot.listof`. See the Examples.

The argument `panel.args` determines extra graphics parameters for each panel. It should be a function that will be called as `panel.args(i)` where `i` is the panel number. Its return value should be a list of graphics parameters that can be passed to the relevant `plot` method. These parameters override any parameters specified in the `...` arguments.

The arguments `panel.begin` and `panel.end` determine graphics that will be plotted before and after each panel is plotted. They may be objects of some class that can be plotted with the generic `plot` command. Alternatively they may be functions that will be called as `panel.begin(i, y, main=main.panel[i])` and `panel.end(i, y, add=TRUE)` where `i` is the panel number and `y = x[[i]]`.

If all entries of `x` are pixel images, the function `image.listof` is called to control the plotting. The arguments `equal.ribbon` and `col` can be used to determine the colour map or maps applied.

If `equal.scales=FALSE` (the default), then the plot panels will have equal height on the plot device (unless there is only one column of panels, in which case they will have equal width on the plot device). This means that the objects are plotted at different physical scales, by default.

If `equal.scales=TRUE`, then the dimensions of the plot panels on the plot device will be proportional to the spatial dimensions of the corresponding components of `x`. This means that the objects will be plotted at *approximately* equal physical scales. If these objects have very different spatial sizes, the plot command could fail (when it tries to plot the smaller objects at a tiny scale), with an error message that the figure margins are too large.

The objects will be plotted at *exactly* equal physical scales, and *exactly* aligned on the device, under the following conditions:

- every component of x is a spatial object whose position can be shifted by shift;
- panel.begin and panel.end are either NULL or they are spatial objects whose position can be shifted by shift;
- adorn.left, adorn.right, adorn.top and adorn.bottom are all NULL.

Another special case is when every component of x is an object of class "fv" representing a function. If equal.scales=TRUE then all these functions will be plotted with the same axis scales (i.e. with the same xlim and the same ylim).

Value

Null.

Spacing between plots

The spacing between individual plots is controlled by the parameters mar.panel, hsep and vsep.

If equal.scales=FALSE, the plot panels are logically separate plots. The margins for each panel are determined by the argument mar.panel which becomes the graphics parameter mar described in the help file for par. One unit of mar corresponds to one line of text in the margin. If hsep or vsep are present, mar.panel is augmented by c(vsep, hsep, vsep, hsep)/2.

If equal.scales=TRUE, all the plot panels are drawn in the same coordinate system which represents a physical scale. The unit of measurement for mar.panel[1,3] is one-sixth of the greatest height of any object plotted in the same row of panels, and the unit for mar.panel[2,4] is one-sixth of the greatest width of any object plotted in the same column of panels. If hsep or vsep are present, they are interpreted in the same units as mar.panel[2] and mar.panel[1] respectively.

Error messages

If the error message 'Figure margins too large' occurs, this generally means that one of the objects had a much smaller physical scale than the others. Ensure that equal.scales=FALSE and increase the values of mar.panel.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

print.listof, contour.listof, image.listof, density.splitppp

Examples

```
# Intensity estimate of multitype point pattern
 plot(D <- density(split(amacrine)))
 plot(D, main="", equal.ribbon=TRUE,
      panel.end=function(i,y,...){contour(y, ...)})

# list of 3D point patterns
 ape1 <- osteo[osteo$shortid==4, "pts", drop=TRUE]
 class(ape1)
```

```
plot(ape1, main.panel="", mar.panel=0.1, hsep=0.7, vsep=1,
     cex=1.5, pch=21, bg='white')
```

---

plot.lpp *Plot Point Pattern on Linear Network*

Description

Plots a point pattern on a linear network. Plot method for the class "lpp" of point patterns on a linear network.

Usage

```
## S3 method for class 'lpp'
plot(x, ..., main, add = FALSE,
                    use.marks=TRUE, which.marks=NULL,
                    show.all = !add, show.window=FALSE, show.network=TRUE,
                    do.plot = TRUE, multiplot=TRUE)
```

Arguments

`x`	Point pattern on a linear network (object of class "lpp").
`...`	Additional arguments passed to `plot.linnet` or `plot.ppp`.
`main`	Main title for plot.
`add`	Logical value indicating whether the plot is to be added to the existing plot (`add=TRUE`) or whether a new plot should be initialised (`add=FALSE`, the default).
`use.marks`	logical flag; if `TRUE`, plot points using a different plotting symbol for each mark; if `FALSE`, only the locations of the points will be plotted, using `points()`.
`which.marks`	Index determining which column of marks to use, if the marks of `x` are a data frame. A character or integer vector identifying one or more columns of marks. If `add=FALSE` then the default is to plot all columns of marks, in a series of separate plots. If `add=TRUE` then only one column of marks can be plotted, and the default is `which.marks=1` indicating the first column of marks.
`show.all`	Logical value indicating whether to plot everything including the main title and the window containing the network.
`show.window`	Logical value indicating whether to plot the window containing the network. Overrides `show.all`.
`show.network`	Logical value indicating whether to plot the network.
`do.plot`	Logical value determining whether to actually perform the plotting.
`multiplot`	Logical value giving permission to display multiple plots.

Details

The linear network is plotted by `plot.linnet`, then the points are plotted by `plot.ppp`.

Commonly-used arguments include:

- `col` and `lwd` for the colour and width of lines in the linear network
- `cols` for the colour or colours of the points

- chars for the plot characters representing different types of points
- legend and leg.side to control the graphics legend

Note that the linear network will be plotted even when add=TRUE, unless show.network=FALSE.

Value

(Invisible) object of class "symbolmap" giving the correspondence between mark values and plotting characters.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

lpp.

See plot.ppp for options for representing the points.

See also points.lpp, text.lpp.

Examples

```
plot(chicago, cols=1:6)
```

---

plot.lppm *Plot a Fitted Point Process Model on a Linear Network*

Description

Plots the fitted intensity of a point process model on a linear network.

Usage

```
## S3 method for class 'lppm'
plot(x, ..., type="trend")
```

Arguments

x	An object of class "lppm" representing a fitted point process model on a linear network.
...	Arguments passed to plot.linim to control the plot.
type	Character string (either "trend" or "cif") determining whether to plot the fitted first order trend or the conditional intensity.

Details

This function is the plot method for the class "lppm". It computes the fitted intensity of the point process model, and displays it using plot.linim.

The default is to display intensity values as colours. Alternatively if the argument style="width" is given, intensity values are displayed as the widths of thick lines drawn over the network.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

lppm, plot.linim, methods.lppm, predict.lppm.

Examples

```
X <- runiflpp(10, simplenet)
fit <- lppm(X ~x)
plot(fit)
plot(fit, style="width")
```

---

plot.mppm *plot a Fitted Multiple Point Process Model*

Description

Given a point process model fitted to multiple point patterns by mppm, compute spatial trend or conditional intensity surface of the model, in a form suitable for plotting, and (optionally) plot this surface.

Usage

```
## S3 method for class 'mppm'
plot(x, ...,
                trend=TRUE, cif=FALSE, se=FALSE,
                how=c("image", "contour", "persp"))
```

Arguments

x	A point process model fitted to multiple point patterns, typically obtained from the model-fitting algorithm mppm. An object of class "mppm".
...	Arguments passed to plot.ppm or plot.anylist controlling the plot.
trend	Logical value indicating whether to plot the fitted trend.
cif	Logical value indicating whether to plot the fitted conditional intensity.
se	Logical value indicating whether to plot the standard error of the fitted trend.
how	Single character string indicating the style of plot to be performed.

Details

This is the plot method for the class "mppm" of point process models fitted to multiple point patterns (see mppm).

It invokes subfits to compute the fitted model for each individual point pattern dataset, then calls plot.ppm to plot these individual models. These individual plots are displayed using plot.anylist, which generates either a series of separate plot frames or an array of plot panels on a single page.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

See Also

plot.ppm, mppm, plot.listof

Examples

```
# Synthetic data from known model
n <- 9
H <- hyperframe(V=1:n,
                U=runif(n, min=-1, max=1))
H$Z <- setcov(square(1))
H$U <- with(H, as.im(U, as.rectangle(Z)))
H$Y <- with(H, rpoispp(eval.im(exp(2+3*Z))))

fit <- mppm(Y ~Z + U + V, data=H)

plot(fit)
```

---

plot.msr *Plot a Signed or Vector-Valued Measure*

Description

Plot a signed measure or vector-valued measure.

Usage

```
  ## S3 method for class 'msr'
plot(x, ...,
                      add = FALSE,
                      how = c("image", "contour", "imagecontour"),
                      main = NULL,
                      do.plot = TRUE,
                      multiplot = TRUE,
                      massthresh = 0,
                      equal.markscale = FALSE,
                      equal.ribbon = FALSE)
```

Arguments

x	The signed or vector measure to be plotted. An object of class "msr" (see msr).
...	Extra arguments passed to Smooth.ppp to control the interpolation of the continuous density component of x, or passed to plot.im or plot.ppp to control the appearance of the plot.
add	Logical flag; if TRUE, the graphics are added to the existing plot. If FALSE (the default) a new plot is initialised.
how	String indicating how to display the continuous density component.
main	String. Main title for the plot.
do.plot	Logical value determining whether to actually perform the plotting.
multiplot	Logical value indicating whether it is permissible to display a plot with multiple panels (representing different components of a vector-valued measure, or different types of points in a multitype measure.)
massthresh	Threshold for plotting atoms. A single numeric value or NULL. If massthresh=0 (the default) then only atoms with nonzero mass will be plotted. If massthresh > 0 then only atoms whose absolute mass exceeds massthresh will be plotted. If massthresh=NULL, then all atoms of the measure will be plotted.
equal.markscale	Logical value indicating whether different panels should use the same symbol map (to represent the masses of atoms of the measure).
equal.ribbon	Logical value indicating whether different panels should use the same colour map (to represent the density values in the diffuse component of the measure).

Details

This is the plot method for the class "msr".

The continuous density component of x is interpolated from the existing data by Smooth.ppp, and then displayed as a colour image by plot.im.

The discrete atomic component of x is then superimposed on this image by plotting the atoms as circles (for positive mass) or squares (for negative mass) by plot.ppp. By default, atoms with zero mass are not plotted at all.

To smooth both the discrete and continuous components, use Smooth.msr.

Use the argument clipwin to restrict the plot to a subset of the full data.

To remove atoms with tiny masses, use the argument massthresh.

Value

(Invisible) colour map (object of class "colourmap") for the colour image.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

msr, Smooth.ppp, Smooth.msr, plot.im, plot.ppp

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
rp <- residuals(fit, type="pearson")
rs <- residuals(fit, type="score")

plot(rp)
plot(rs)
plot(rs, how="contour")
```

---

plot.onearrow *Plot an Arrow*

Description

Plots an object of class "onearrow".

Usage

```
## S3 method for class 'onearrow'
plot(x, ...,
  add = FALSE, main = "",
  retract = 0.05, headfraction = 0.25, headangle = 12, headnick = 0.1,
  col.head = NA, lwd.head = lwd, lwd = 1, col = 1,
  zap = FALSE, zapfraction = 0.07,
  pch = 1, cex = 1, do.plot = TRUE, do.points = FALSE, show.all = !add)
```

Arguments

x	Object of class "onearrow" to be plotted. This object is created by the command onearrow.
...	Additional graphics arguments passed to segments to control the appearance of the line.
add	Logical value indicating whether to add graphics to the existing plot (add=TRUE) or to start a new plot (add=FALSE).
main	Main title for the plot.
retract	Fraction of length of arrow to remove at each end.
headfraction	Length of arrow head as a fraction of overall length of arrow.
headangle	Angle (in degrees) between the outer edge of the arrow head and the shaft of the arrow.
headnick	Size of the nick in the trailing edge of the arrow head as a fraction of length of arrow head.
col.head,lwd.head	Colour and line style of the filled arrow head.
col,lwd	Colour and line style of the arrow shaft.
zap	Logical value indicating whether the arrow should include a Z-shaped (lightning-bolt) feature in the middle of the shaft.
zapfraction	Size of Z-shaped deviation as a fraction of total arrow length.

pch,cex	Plot character and character size for the two end points of the arrow, if do.points=TRUE.
do.plot	Logical. Whether to actually perform the plot.
do.points	Logical. Whether to display the two end points of the arrow as well.
show.all	Internal use only.

Details

The argument x should be an object of class "onearrow" created by the command onearrow.

Value

A window (class "owin") enclosing the plotted graphics.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

onearrow, yardstick

Examples

```
oa <- onearrow(cells[c(1, 42)])
plot(oa)
plot(oa, zap=TRUE, do.points=TRUE, col.head="pink", col="red")
```

---

plot.owin *Plot a Spatial Window*

Description

Plot a two-dimensional window of observation for a spatial point pattern

Usage

```
 ## S3 method for class 'owin'
plot(x, main, add=FALSE, ..., box, edge=0.04,
                      type=c("w","n"), show.all=!add,
                      hatch=FALSE,
                      hatchargs=list(),
                      invert=FALSE, do.plot=TRUE,
                      claim.title.space=FALSE)
```

Arguments

x	The window to be plotted. An object of class owin, or data which can be converted into this format by as.owin().
main	text to be displayed as a title above the plot.
add	logical flag: if TRUE, draw the window in the current plot; if FALSE, generate a new plot.
...	extra arguments controlling the appearance of the plot. These arguments are passed to polygon if x is a polygonal or rectangular window, or passed to image.default if x is a binary mask. See Details.
box	logical flag; if TRUE, plot the enclosing rectangular box
edge	nonnegative number; the plotting region will have coordinate limits that are 1 + edge times as large as the limits of the rectangular box that encloses the pattern.
type	Type of plot: either "w" or "n". If type="w" (the default), the window is plotted. If type="n" and add=TRUE, a new plot is initialised and the coordinate system is established, but nothing is drawn.
show.all	Logical value indicating whether to plot everything including the main title.
hatch	logical flag; if TRUE, the interior of the window will be shaded by texture, such as a grid of parallel lines.
hatchargs	List of arguments passed to add.texture to control the texture shading when hatch=TRUE.
invert	logical flag; when the window is a binary pixel mask, the mask colours will be inverted if invert=TRUE.
do.plot	Logical value indicating whether to actually perform the plot.
claim.title.space	Logical value indicating whether extra space for the main title should be allocated when declaring the plot dimensions. Should be set to FALSE under normal conditions.

Details

This is the plot method for the class owin. The action is to plot the boundary of the window on the current plot device, using equal scales on the x and y axes.

If the window x is of type "rectangle" or "polygonal", the boundary of the window is plotted as a polygon or series of polygons. If x is of type "mask" the discrete raster approximation of the window is displayed as a binary image (white inside the window, black outside).

Graphical parameters controlling the display (e.g. setting the colours) may be passed directly via the ... arguments, or indirectly reset using spatstat.options.

When x is of type "rectangle" or "polygonal", it is plotted by the R function polygon. To control the appearance (colour, fill density, line density etc) of the polygon plot, determine the required argument of polygon and pass it through ... For example, to paint the interior of the polygon in red, use the argument col="red". To draw the polygon edges in green, use border="green". To suppress the drawing of polygon edges, use border=NA.

When x is of type "mask", it is plotted by image.default. The appearance of the image plot can be controlled by passing arguments to image.default through .... The default appearance can also be changed by setting the parameter par.binary of spatstat.options.

To zoom in (to view only a subset of the window at higher magnification), use the graphical arguments xlim and ylim to specify the desired rectangular field of view. (The actual field of view may be larger, depending on the graphics device).

Value

none.

Notes on Filled Polygons with Holes

The function `polygon` can only handle polygons without holes. To plot polygons with holes in a solid colour, we have implemented two workarounds.

polypath function: The first workaround uses the relatively new function `polypath` which *does* have the capability to handle polygons with holes. However, not all graphics devices support `polypath`. The older devices `xfig` and `pictex` do not support `polypath`. On a Windows system, the default graphics device `windows` supports `polypath`. On a Linux system, the default graphics device `X11(type="Xlib")` does *not* support `polypath` but `X11(type="cairo")` does support it. See `X11` and the section on Cairo below.

polygon decomposition: The other workaround involves decomposing the polygonal window into pieces which do not have holes. This code is experimental but works in all our test cases. If this code fails, a warning will be issued, and the filled colours will not be plotted.

Cairo graphics on a Linux system

Linux systems support the graphics device `X11(type="cairo")` (see `X11`) provided the external library **cairo** is installed on the computer. See `www.cairographics.org` for instructions on obtaining and installing **cairo**. After having installed **cairo** one needs to re-install R from source so that it has **cairo** capabilites. To check whether your current installation of R has **cairo** capabilities, type (in R) `capabilities()["cairo"]`. The default type for `X11` is controlled by `X11.options`. You may find it convenient to make **cairo** the default, e.g. via your `.Rprofile`. The magic incantation to put into `.Rprofile` is

```
setHook(packageEvent("graphics", "onLoad"),
function(...) grDevices::X11.options(type="cairo"))
```

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`owin.object`, `plot.ppp`, `polygon`, `image.default`, `spatstat.options`

Examples

```
# rectangular window
 plot(Window(nztrees))
 abline(v=148, lty=2)

# polygonal window
w <- Window(demopat)
plot(w)
plot(w, col="red", border="green", lwd=2)
plot(w, hatch=TRUE, lwd=2)

# binary mask
```

```
we <- as.mask(w)
plot(we)
op <- spatstat.options(par.binary=list(col=grey(c(0.5,1))))
plot(we)
spatstat.options(op)
```

`plot.plotppm` *Plot a plotppm Object Created by plot.ppm*

Description

The function plot.ppm produces objects which specify plots of fitted point process models. The function plot.plotppm carries out the actual plotting of these objects.

Usage

```
## S3 method for class 'plotppm'
plot(x, data = NULL, trend = TRUE, cif = TRUE,
              se = TRUE, pause = interactive(),
              how = c("persp", "image", "contour"),
              ..., pppargs)
```

Arguments

`x`	An object of class `plotppm` produced by `plot.ppm()` .
`data`	The point pattern (an object of class ppp) to which the point process model was fitted (by `ppm`).
`trend`	Logical scalar; should the trend component of the fitted model be plotted?
`cif`	Logical scalar; should the complete conditional intensity of the fitted model be plotted?
`se`	Logical scalar; should the estimated standard error of the fitted intensity be plotted?
`pause`	Logical scalar indicating whether to pause with a prompt after each plot. Set `pause=FALSE` if plotting to a file.
`how`	Character string or character vector indicating the style or styles of plots to be performed.
`...`	Extra arguments to the plotting functions `persp`, `image` and `contour`.
`pppargs`	List of extra arguments passed to `plot.ppp` when displaying the original point pattern data.

Details

If argument `data` is supplied then the point pattern will be superimposed on the image and contour plots.

Sometimes a fitted model does not have a trend component, or the trend component may constitute all of the conditional intensity (if the model is Poisson). In such cases the object `x` will not contain a trend component, or will contain only a trend component. This will also be the case if one of the arguments `trend` and `cif` was set equal to `FALSE` in the call to `plot.ppm()` which produced `x`. If this is so then only the item which is present will be plotted. Explicitly setting `trend=TRUE`, or `cif=TRUE`, respectively, will then give an error.

Value

None.

Warning

Arguments which are passed to `persp`, `image`, and `contour` via the ... argument get passed to any of the other functions listed in the `how` argument, and won't be recognized by them. This leads to a lot of annoying but harmless warning messages. Arguments to `persp` may be supplied via `spatstat.options()` which alleviates the warning messages in this instance.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`plot.ppm()`

Examples

```
## Not run:
m <- ppm(cells ~ 1, Strauss(0.05))
mpic <- plot(m)
# Perspective plot only, with altered parameters:
 plot(mpic,how="persp", theta=-30,phi=40,d=4)
# All plots, with altered parameters for perspective plot:
op <- spatstat.options(par.persp=list(theta=-30,phi=40,d=4))
plot(mpic)
# Revert
spatstat.options(op)

## End(Not run)
```

`plot.pp3` *Plot a Three-Dimensional Point Pattern*

Description

Plots a three-dimensional point pattern.

Usage

```
## S3 method for class 'pp3'
plot(x, ..., eye=NULL, org=NULL, theta=25, phi=15,
                    type=c("p", "n", "h"),
                    box.back=list(col="pink"),
                    box.front=list(col="blue", lwd=2))
```

Arguments

- `x` Three-dimensional point pattern (object of class "pp3").
- `...` Arguments passed to points controlling the appearance of the points.
- `eye` Optional. Eye position. A numeric vector of length 3 giving the location from which the scene is viewed.
- `org` Optional. Origin (centre) of the view. A numeric vector of length 3 which will be at the centre of the view.
- `theta,phi` Optional angular coordinates (in degrees) specifying the direction from which the scene is viewed: theta is the azimuth and phi is the colatitude. Ignored if eye is given.
- `type` Type of plot: type="p" for points, type="h" for points on vertical lines, type="n" for box only.
- `box.front,box.back` How to plot the three-dimensional box that contains the points. A list of graphical arguments passed to segments, or a logical value indicating whether or not to plot the relevant part of the box. See Details.

Details

This is the plot method for objects of class "pp3". It generates a two-dimensional plot of the point pattern x and its containing box as if they had been viewed from the location specified by eye (or from the direction specified by theta and phi).

The edges of the box at the 'back' of the scene (as viewed from the eye position) are plotted first. Then the points are added. Finally the remaining 'front' edges are plotted. The arguments box.back and box.front specify graphical parameters for drawing the back and front edges, respectively. Alternatively box.back=FALSE specifies that the back edges shall not be drawn.

Note that default values of arguments to plot.pp3 can be set by spatstat.options("par.pp3").

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

pp3, spatstat.options.

Examples

```
X <- osteo$pts[[1]]
plot(X, main="Osteocyte lacunae, animal 1, brick 1",
     cex=1.5, pch=16)
plot(X, main="", box.back=list(lty=3))
```

plot.ppm *plot a Fitted Point Process Model*

Description

Given a fitted point process model obtained by ppm, create spatial trend and conditional intensity surfaces of the model, in a form suitable for plotting, and (optionally) plot these surfaces.

Usage

```
## S3 method for class 'ppm'
plot(x, ngrid = c(40,40), superimpose = TRUE,
                  trend = TRUE, cif = TRUE, se = TRUE, pause = interactive(),
                  how=c("persp","image", "contour"), plot.it = TRUE,
                  locations = NULL, covariates=NULL, ...)
```

Arguments

x	A fitted point process model, typically obtained from the model-fitting algorithm ppm. An object of class "ppm".
ngrid	The dimensions for a grid on which to evaluate, for plotting, the spatial trend and conditional intensity. A vector of 1 or 2 integers. If it is of length 1, ngrid is replaced by c(ngrid,ngrid).
superimpose	logical flag; if TRUE (and if plot=TRUE) the original data point pattern will be superimposed on the plots.
trend	logical flag; if TRUE, the spatial trend surface will be produced.
cif	logical flag; if TRUE, the conditional intensity surface will be produced.
se	logical flag; if TRUE, the estimated standard error of the spatial trend surface will be produced.
pause	logical flag indicating whether to pause with a prompt after each plot. Set pause=FALSE if plotting to a file. (This flag is ignored if plot=FALSE).
how	character string or character vector indicating the style or styles of plots to be performed. Ignored if plot=FALSE.
plot.it	logical scalar; should a plot be produced immediately?
locations	If present, this determines the locations of the pixels at which predictions are computed. It must be a binary pixel image (an object of class "owin" with type "mask"). (Incompatible with ngrid).
covariates	Values of external covariates required by the fitted model. Passed to predict.ppm.
...	extra arguments to the plotting functions persp, image and contour.

Details

This is the plot method for the class "ppm" (see ppm.object for details of this class).

It invokes predict.ppm to compute the spatial trend and conditional intensity of the fitted point process model. See predict.ppm for more explanation about spatial trend and conditional intensity.

The default action is to create a rectangular grid of points in (the bounding box of) the observation window of the data point pattern, and evaluate the spatial trend and conditional intensity of the fitted spatial point process model x at these locations. If the argument locations= is supplied, then the spatial trend and conditional intensity are calculated at the grid of points specified by this argument.

The argument locations, if present, should be a binary image mask (an object of class "owin" and type "mask"). This determines a rectangular grid of locations, or a subset of such a grid, at which predictions will be computed. Binary image masks are conveniently created using as.mask.

The argument covariates gives the values of any spatial covariates at the prediction locations. If the trend formula in the fitted model involves spatial covariates (other than the Cartesian coordinates x, y) then covariates is required.

The argument covariates has the same format and interpretation as in predict.ppm. It may be either a data frame (the number of whose rows must match the number of pixels in locations multiplied by the number of possible marks in the point pattern), or a list of images. If argument locations is not supplied, and covariates **is** supplied, then it **must** be a list of images.

If the fitted model was a marked (multitype) point process, then predictions are made for each possible mark value in turn.

If the fitted model had no spatial trend, then the default is to omit calculating this (flat) surface, unless trend=TRUE is set explicitly.

If the fitted model was Poisson, so that there were no spatial interactions, then the conditional intensity and spatial trend are identical, and the default is to omit the conditional intensity, unless cif=TRUE is set explicitly.

If plot.it=TRUE then plot.plotppm() is called upon to plot the class plotppm object which is produced. (That object is also returned, silently.)

Plots are produced successively using persp, image and contour (or only a selection of these three, if how is given). Extra graphical parameters controlling the display may be passed directly via the arguments ... or indirectly reset using spatstat.options.

Value

An object of class plotppm. Such objects may be plotted by plot.plotppm().

This is a list with components named trend and cif, either of which may be missing. They will be missing if the corresponding component does not make sense for the model, or if the corresponding argument was set equal to FALSE.

Both trend and cif are lists of images. If the model is an unmarked point process, then they are lists of length 1, so that trend[[1]] is an image of the spatial trend and cif[[1]] is an image of the conditional intensity.

If the model is a marked point process, then trend[[i]] is an image of the spatial trend for the mark m[i], and cif[[1]] is an image of the conditional intensity for the mark m[i], where m is the vector of levels of the marks.

Warnings

See warnings in predict.ppm.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

plot.plotppm, ppm, ppm.object, predict.ppm, print.ppm, persp, image, contour, plot, spatstat.options

Examples

```
m <- ppm(cells ~1, Strauss(0.05))
pm <- plot(m) # The object ``pm'' will be plotted as well as saved
              # for future plotting.
```

---

plot.ppp *plot a Spatial Point Pattern*

Description

Plot a two-dimensional spatial point pattern

Usage

```
## S3 method for class 'ppp'
plot(x, main, ..., clipwin=NULL,
                    chars=NULL, cols=NULL,
                    use.marks=TRUE, which.marks=NULL,
                    add=FALSE, type=c("p","n"),
                    legend=TRUE,
                    leg.side=c("left", "bottom", "top", "right"),
                    leg.args=list(),
                    symap=NULL, maxsize=NULL, meansize=NULL, markscale=NULL,
                    zap=0.01,
                    show.window=show.all, show.all=!add, do.plot=TRUE,
                    multiplot=TRUE)
```

Arguments

`x`	The spatial point pattern to be plotted. An object of class "ppp", or data which can be converted into this format by `as.ppp()`.
`main`	text to be displayed as a title above the plot.
`...`	extra arguments that will be passed to the plotting functions `plot.default`, `points` and/or `symbols`.
`clipwin`	Optional. A window (object of class "owin"). Only this subset of the image will be displayed.
`chars`	plotting character(s) used to plot points.
`cols`	the colour(s) used to plot points.
`use.marks`	logical flag; if `TRUE`, plot points using a different plotting symbol for each mark; if `FALSE`, only the locations of the points will be plotted, using `points()`.
`which.marks`	Index determining which column of marks to use, if the marks of `x` are a data frame. A character or integer vector identifying one or more columns of marks. If `add=FALSE` then the default is to plot all columns of marks, in a series of separate plots. If `add=TRUE` then only one column of marks can be plotted, and the default is `which.marks=1` indicating the first column of marks.
`add`	logical flag; if `TRUE`, just the points are plotted, over the existing plot. A new plot is not created, and the window is not plotted.

type	Type of plot: either "p" or "n". If type="p" (the default), both the points and the observation window are plotted. If type="n", only the window is plotted.
legend	Logical value indicating whether to add a legend showing the mapping between mark values and graphical symbols (for a marked point pattern).
leg.side	Position of legend relative to main plot.
leg.args	List of additional arguments passed to plot.symbolmap or symbolmap to control the legend. In addition to arguments documented under plot.symbolmap, and graphical arguments recognised by symbolmap, the list may also include the argument sep giving the separation between the main plot and the legend, or sep.frac giving the separation as a fraction of the relevant dimension (width or height) of the main plot.
symap	Optional. The graphical symbol map to be applied to the marks. An object of class "symbolmap"; see symbolmap.
maxsize	*Maximum* physical size of the circles/squares plotted when x is a marked point pattern with numerical marks. Incompatible with meansize and markscale. Ignored if symap is given.
meansize	*Average* physical size of the circles/squares plotted when x is a marked point pattern with numerical marks. Incompatible with maxsize and markscale. Ignored if symap is given.
markscale	physical scale factor determining the sizes of the circles/squares plotted when x is a marked point pattern with numerical marks. Mark value will be multiplied by markscale to determine physical size. Incompatible with maxsize and meansize. Ignored if symap is given.
zap	Fraction between 0 and 1. When x is a marked point pattern with numerical marks, zap is the smallest mark value (expressed as a fraction of the maximum possible mark) that will be plotted. Any points which have marks smaller in absolute value than zap * max(abs(marks(x))) will not be plotted.
show.window	Logical value indicating whether to plot the observation window of x.
show.all	Logical value indicating whether to plot everything including the main title and the observation window of x.
do.plot	Logical value determining whether to actually perform the plotting.
multiplot	Logical value giving permission to display multiple plots.

Details

This is the plot method for point pattern datasets (of class "ppp", see ppp.object).

First the observation window Window(x) is plotted (if show.window=TRUE). Then the points themselves are plotted, in a fashion that depends on their marks, as follows.

unmarked point pattern: If the point pattern does not have marks, or if use.marks = FALSE, then the locations of all points will be plotted using a single plot character

multitype point pattern: If x$marks is a factor, then each level of the factor is represented by a different plot character.

continuous marks: If x$marks is a numeric vector, the marks are rescaled to the unit interval and each point is represented by a circle with *diameter* proportional to the rescaled mark (if the value is positive) or a square with *side length* proportional to the absolute value of the rescaled mark (if the value is negative).

other kinds of marks: If x$marks is neither numeric nor a factor, then each possible mark will be represented by a different plotting character. The default is to represent the ith smallest mark value by points(..., pch=i).

If there are several columns of marks, and if which.marks is missing or NULL, then

- if add=FALSE and multiplot=TRUE the default is to plot all columns of marks, in a series of separate plots, placed side-by-side. The plotting is coordinated by plot.listof, which calls plot.ppp to make each of the individual plots.
- Otherwise, only one column of marks can be plotted, and the default is which.marks=1 indicating the first column of marks.

Plotting of the window Window(x) is performed by plot.owin. This plot may be modified through the ... arguments. In particular the extra argument border determines the colour of the window, if the window is not a binary mask.

Plotting of the points themselves is performed by the function points, except for the case of continuous marks, where it is performed by symbols. Their plotting behaviour may be modified through the ... arguments.

The argument chars determines the plotting character or characters used to display the points (in all cases except for the case of continuous marks). For an unmarked point pattern, this should be a single integer or character determining a plotting character (see par("pch")). For a multitype point pattern, chars should be a vector of integers or characters, of the same length as levels(x$marks), and then the ith level or type will be plotted using character chars[i].

If chars is absent, but there is an extra argument pch, then this will determine the plotting character for all points.

The argument cols determines the colour or colours used to display the points. For an unmarked point pattern, cols should be a character string determining a colour. For a multitype point pattern, cols should be a character vector, of the same length as levels(marks(x)): that is, there is one colour for each possible mark value. The ith level or type will be plotted using colour cols[i]. For a point pattern with continuous marks, cols can be either a character string or a character vector specifying colour values: the range of mark values will be mapped to the specified colours.

If cols is absent, the colours used to plot the points may be determined by the extra argument fg (for multitype point patterns) or the extra argument col (for all other cases). Note that specifying col will also apply this colour to the window itself.

The default colour for the points is a semi-transparent grey, if this is supported by the plot device. This behaviour can be suppressed (so that the default colour is non-transparent) by setting spatstat.options(transparent=FALSE).

The arguments maxsize, meansize and markscale incompatible. They control the physical size of the circles and squares which represent the marks in a point pattern with continuous marks. The size of a circle is defined as its *diameter*; the size of a square is its side length. If markscale is given, then a mark value of m is plotted as a circle of diameter m * markscale (if m is positive) or a square of side abs(m) * markscale (if m is negative). If maxsize is given, then the largest mark in absolute value, mmax=max(abs(marks(x))), will be scaled to have physical size maxsize. If meansize is given, then the average absolute mark value, mmean=mean(abs(marks(x))), will be scaled to have physical size meansize.

The user can set the default values of these plotting parameters using spatstat.options("par.points").

To zoom in (to view only a subset of the point pattern at higher magnification), use the graphical arguments xlim and ylim to specify the rectangular field of view.

The value returned by this plot function is an object of class "symbolmap" representing the mapping from mark values to graphical symbols. See symbolmap. It can be used to make a suitable legend, or to ensure that two plots use the same graphics map.

Value

(Invisible) object of class "symbolmap" giving the correspondence between mark values and plotting characters.

Removing White Space Around The Plot

A frequently-asked question is: How do I remove the white space around the plot? Currently plot.ppp uses the base graphics system of R, so the space around the plot is controlled by parameters to par. To reduce the white space, change the parameter mar. Typically, par(mar=rep(0.5, 4)) is adequate, if there are no annotations or titles outside the window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

iplot, ppp.object, plot, par, points, text.ppp, plot.owin, symbols

Examples

```
plot(cells)

plot(cells, pch=16)

# make the plotting symbols larger (for publication at reduced scale)
plot(cells, cex=2)

# set it in spatstat.options
oldopt <- spatstat.options(par.points=list(cex=2))
plot(cells)
spatstat.options(oldopt)

# multitype
plot(lansing)

# marked by a real number
plot(longleaf)

# just plot the points
plot(longleaf, use.marks=FALSE)
plot(unmark(longleaf)) # equivalent

# point pattern with multiple marks
plot(finpines)
plot(finpines, which.marks="height")

# controlling COLOURS of points
plot(cells, cols="blue")
plot(lansing, cols=c("black", "yellow", "green",
                     "blue","red","pink"))
plot(longleaf, fg="blue")

# make window purple
plot(lansing, border="purple")
```

```
# make everything purple
plot(lansing, border="purple", cols="purple", col.main="purple",
              leg.args=list(col.axis="purple"))

# controlling PLOT CHARACTERS for multitype pattern
plot(lansing, chars = 11:16)
plot(lansing, chars = c("o","h","m",".","o","o"))

## multitype pattern mapped to symbols
plot(amacrine, shape=c("circles", "squares"), size=0.04)
plot(amacrine, shape="arrows", direction=c(0,90), size=0.07)

## plot trees as trees!
plot(lansing, shape="arrows", direction=90, cols=1:6)

# controlling MARK SCALE for pattern with numeric marks
plot(longleaf, markscale=0.1)
plot(longleaf, maxsize=5)
plot(longleaf, meansize=2)

# draw circles of diameter equal to nearest neighbour distance
plot(cells %mark% nndist(cells), markscale=1, legend=FALSE)

# inspecting the symbol map
v <- plot(amacrine)
v

## variable colours ('cols' not 'col')
plot(longleaf, cols=function(x) ifelse(x < 30, "red", "black"))

## re-using the same mark scale
a <- plot(longleaf)
juveniles <- longleaf[marks(longleaf) < 30]
plot(juveniles, symap=a)

## numerical marks mapped to symbols of fixed size with variable colour
ra <- range(marks(longleaf))
colmap <- colourmap(terrain.colors(20), range=ra)
## filled plot characters are the codes 21-25
## fill colour is indicated by 'bg'
sy <- symbolmap(pch=21, bg=colmap, range=ra)
plot(longleaf, symap=sy)

## or more compactly..
plot(longleaf, bg=terrain.colors(20), pch=21, cex=1)

## clipping
plot(humberside)
B <- owin(c(4810, 5190), c(4180, 4430))
plot(B, add=TRUE, border="red")
plot(humberside, clipwin=B, main="Humberside (clipped)")
```

---

plot.psp *plot a Spatial Line Segment Pattern*

Description

Plot a two-dimensional line segment pattern

Usage

```
## S3 method for class 'psp'
plot(x, ..., main, add=FALSE,
                     show.all=!add, show.window=show.all,
                     which.marks=1, ribbon=show.all,
                     ribsep=0.15, ribwid=0.05, ribn=1024,
                     do.plot=TRUE)
```

Arguments

`x`	The line segment pattern to be plotted. An object of class "psp", or data which can be converted into this format by `as.psp()`.
`...`	extra arguments that will be passed to the plotting functions `segments` (to plot the segments) and `plot.owin` (to plot the observation window).
`main`	Character string giving a title for the plot.
`add`	Logical. If `TRUE`, the current plot is not erased; the segments are plotted on top of the current plot, and the window is not plotted (by default).
`show.all`	Logical value specifying whether to plot everything including the window, main title, and colour ribbon.
`show.window`	Logical value specifying whether to plot the window.
`which.marks`	Index determining which column of marks to use, if the marks of `x` are a data frame. A character string or an integer. Defaults to 1 indicating the first column of marks.
`ribbon`	Logical flag indicating whether to display a ribbon showing the colour map (in which mark values are associated with colours).
`ribsep`	Factor controlling the space between the ribbon and the image.
`ribwid`	Factor controlling the width of the ribbon.
`ribn`	Number of different values to display in the ribbon.
`do.plot`	Logical value indicating whether to actually perform the plot.

Details

This is the `plot` method for line segment pattern datasets (of class "psp", see `psp.object`). It plots both the observation window `Window(x)` and the line segments themselves.

Plotting of the window `Window(x)` is performed by `plot.owin`. This plot may be modified through the `...` arguments.

Plotting of the segments themselves is performed by the standard R function `segments`. Its plotting behaviour may also be modified through the `...` arguments.

For a *marked* line segment pattern (i.e. if `marks(x)` is not `NULL`) the line segments are plotted in colours determined by the mark values. If `marks(x)` is a data frame, the default is to use the first column of `marks(x)` to determine the colours. To specify another column, use the argument `which.marks`. The colour map (associating mark values with colours) will be displayed as a vertical colour ribbon to the right of the plot, if `ribbon=TRUE`.

Value

(Invisibly) a colour map object specifying the association between marks and colours, if any. The return value also has an attribute "bbox" giving a bounding box for the plot.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

psp.object, plot, par, plot.owin, text.psp, symbols

Examples

```
X <- psp(runif(20), runif(20), runif(20), runif(20), window=owin())
plot(X)
plot(X, lwd=3)
lettuce <- sample(letters[1:4], 20, replace=TRUE)
marks(X) <- data.frame(A=1:20, B=factor(lettuce))
plot(X)
plot(X, which.marks="B")
```

---

plot.quad *Plot a Spatial Quadrature Scheme*

Description

Plot a two-dimensional spatial quadrature scheme.

Usage

```
  ## S3 method for class 'quad'
plot(x, ..., main, add=FALSE, dum=list(), tiles=FALSE)
```

Arguments

x	The spatial quadrature scheme to be plotted. An object of class "quad".
...	extra arguments controlling the plotting of the data points of the quadrature scheme.
main	text to be displayed as a title above the plot.
add	Logical value indicating whether the graphics should be added to the current plot if there is one (add=TRUE) or whether a new plot should be initialised (add=FALSE, the default).
dum	list of extra arguments controlling the plotting of the dummy points of the quadrature scheme. See below.
tiles	Logical value indicating whether to display the tiles used to compute the quadrature weights.

Details

This is the `plot` method for quadrature schemes (objects of class `"quad"`, see `quad.object`).

First the data points of the quadrature scheme are plotted (in their observation window) using `plot.ppp` with any arguments specified in `...`

Then the dummy points of the quadrature scheme are plotted using `plot.ppp` with any arguments specified in `dum`.

By default the dummy points are superimposed onto the plot of data points. This can be overridden by including the argument `add=FALSE` in the list `dum` as shown in the examples. In this case the data and dummy point patterns are plotted separately.

See `par` and `plot.ppp` for other possible arguments controlling the plots.

Value

`NULL`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`quad.object`, `plot.ppp`, `par`

Examples

```
data(nztrees)
Q <- quadscheme(nztrees)

plot(Q, main="NZ trees: quadrature scheme")

oldpar <- par(mfrow=c(2,1))
plot(Q, main="NZ trees", dum=list(add=FALSE))
par(oldpar)
```

`plot.quadratcount` *Plot Quadrat Counts*

Description

Given a table of quadrat counts for a spatial point pattern, plot the quadrats which were used, and display the quadrat count as text in the centre of each quadrat.

Usage

```
## S3 method for class 'quadratcount'
plot(x, ..., add = FALSE,
              entries = as.vector(t(as.table(x))),
              dx = 0, dy = 0, show.tiles = TRUE,
              textargs = list())
```

Arguments

`x`	Object of class `"quadratcount"` produced by the function `quadratcount`.
`...`	Additional arguments passed to `plot.tess` to plot the quadrats.
`add`	Logical. Whether to add the graphics to an existing plot.
`entries`	Vector of numbers to be plotted in each quadrat. The default is to plot the quadrat counts.
`dx,dy`	Horizontal and vertical displacement of text relative to centroid of quadrat.
`show.tiles`	Logical value indicating whether to plot the quadrats.
`textargs`	List containing extra arguments passed to `text.default` to control the annotation.

Details

This is the plot method for the objects of class `"quadratcount"` that are produced by the function `quadratcount`. Given a spatial point pattern, `quadratcount` divides the observation window into disjoint tiles or quadrats, counts the number of points in each quadrat, and stores the result as a contingency table which also belongs to the class `"quadratcount"`.

First the quadrats are plotted (provided `show.tiles=TRUE`, the default). This display can be controlled by passing additional arguments `...` to `plot.tess`.

Then the quadrat counts are printed using `text.default`. This display can be controlled using the arguments `dx,dy` and `textargs`.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`quadratcount`, `plot.tess`, `text.default`, `plot.quadrattest`.

Examples

```
plot(quadratcount(swedishpines, 5))
```

plot.quadrattest *Display the result of a quadrat counting test.*

Description

Given the result of a quadrat counting test, graphically display the quadrats that were used, the observed and expected counts, and the residual in each quadrat.

Usage

```
## S3 method for class 'quadrattest'
plot(x, ..., textargs=list())
```

Arguments

x	Object of class "quadrattest" containing the result of quadrat.test.
...	Additional arguments passed to plot.tess to control the display of the quadrats.
textargs	List of additional arguments passed to text.default to control the appearance of the text.

Details

This is the plot method for objects of class "quadrattest". Such an object is produced by quadrat.test and represents the result of a χ^2 test for a spatial point pattern.

The quadrats are first plotted using plot.tess. Then in each quadrat, the observed and expected counts and the Pearson residual are displayed as text using text.default. Observed count is displayed at top left; expected count at top right; and Pearson residual at bottom.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quadrat.test, plot.tess, text.default, plot.quadratcount

Examples

```
plot(quadrat.test(swedishpines, 3))
```

plot.rppm *Plot a Recursively Partitioned Point Process Model*

Description

Given a model which has been fitted to point pattern data by recursive partitioning, plot the partition tree or the fitted intensity.

Usage

```
## S3 method for class 'rppm'
plot(x, ..., what = c("tree", "spatial"), treeplot=NULL)
```

Arguments

x	Fitted point process model of class "rppm" produced by the function rppm.
what	Character string (partially matched) specifying whether to plot the partition tree or the fitted intensity.
...	Arguments passed to plot.rpart and text.rpart (if what="tree") or passed to plot.im (if what="spatial") controlling the appearance of the plot.
treeplot	Optional. A function to be used to plot and label the partition tree, replacing the two functions plot.rpart and text.rpart.

Details

If what="tree" (the default), the partition tree will be plotted using plot.rpart, and labelled using text.rpart.

If the argument treeplot is given, then plotting and labelling will be performed by treeplot instead. A good choice is the function prp in package **rpart.plot**.

If what="spatial", the predicted intensity will be computed using predict.rppm, and this intensity will be plotted as an image using plot.im.

Value

If what="tree", a list containing x and y coordinates of the plotted nodes of the tree. If what="spatial", the return value of plot.im.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

rppm

Examples

```
# Murchison gold data
mur <- solapply(murchison, rescale, s=1000, unitname="km")
mur$dfault <- distfun(mur$faults)
#
fit <- rppm(gold ~ dfault + greenstone, data=mur)
#
opa <- par(mfrow=c(1,2))
plot(fit)
plot(fit, what="spatial")
par(opa)
```

`plot.scan.test` *Plot Result of Scan Test*

Description

Computes or plots an image showing the likelihood ratio test statistic for the scan test, or the optimal circle radius.

Usage

```
 ## S3 method for class 'scan.test'
plot(x, ..., what=c("statistic", "radius"),
               do.window = TRUE)

 ## S3 method for class 'scan.test'
as.im(X, ..., what=c("statistic", "radius"))
```

Arguments

`x,X`	Result of a scan test. An object of class `"scan.test"` produced by `scan.test`.
`...`	Arguments passed to `plot.im` to control the appearance of the plot.
`what`	Character string indicating whether to produce an image of the (profile) likelihood ratio test statistic (`what="statistic"`, the default) or an image of the optimal value of circle radius (`what="radius"`).
`do.window`	Logical value indicating whether to plot the original window of the data as well.

Details

These functions extract, and plot, the spatially-varying value of the likelihood ratio test statistic which forms the basis of the scan test.

If the test result `X` was based on circles of the same radius `r`, then `as.im(X)` is a pixel image of the likelihood ratio test statistic as a function of the position of the centre of the circle.

If the test result `X` was based on circles of several different radii `r`, then `as.im(X)` is a pixel image of the profile (maximum value over all radii `r`) likelihood ratio test statistic as a function of the position of the centre of the circle, and `as.im(X, what="radius")` is a pixel image giving for each location u the value of `r` which maximised the likelihood ratio test statistic at that location.

The `plot` method plots the corresponding image.

Value

The value of as.im.scan.test is a pixel image (object of class "im"). The value of plot.scan.test is NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

scan.test, scanLRTS

Examples

```
if(interactive()) {
  a <- scan.test(redwood, seq(0.04, 0.1, by=0.01),
                 method="poisson", nsim=19)
} else {
  a <- scan.test(redwood, c(0.05, 0.1), method="poisson", nsim=2)
}
plot(a)
as.im(a)
plot(a, what="radius")
```

---

plot.slrm *Plot a Fitted Spatial Logistic Regression*

Description

Plots a fitted Spatial Logistic Regression model.

Usage

```
## S3 method for class 'slrm'
plot(x, ..., type = "intensity")
```

Arguments

x	a fitted spatial logistic regression model. An object of class "slrm".
...	Extra arguments passed to plot.im to control the appearance of the plot.
type	Character string (partially) matching one of "probabilities", "intensity" or "link".

Details

This is a method for plot for fitted spatial logistic regression models (objects of class "slrm", usually obtained from the function slrm).

This function plots the result of predict.slrm.

Value

None.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

slrm, predict.slrm, plot.im

Examples

```
data(copper)
X <- copper$SouthPoints
Y <- copper$SouthLines
Z <- distmap(Y)
fit <- slrm(X ~ Z)
plot(fit)
plot(fit, type="link")
```

---

plot.solist *Plot a List of Spatial Objects*

Description

Plots a list of two-dimensional spatial objects.

Usage

```
## S3 method for class 'solist'
plot(x, ..., main, arrange=TRUE,
   nrows=NULL, ncols=NULL, main.panel=NULL,
   mar.panel=c(2,1,1,2), hsep=0, vsep=0,
   panel.begin=NULL, panel.end=NULL, panel.args=NULL,
   panel.begin.args=NULL, panel.end.args=NULL, panel.vpad = 0.2,
   plotcommand="plot",
   adorn.left=NULL, adorn.right=NULL, adorn.top=NULL, adorn.bottom=NULL,
   adorn.size=0.2, equal.scales=FALSE, halign=FALSE, valign=FALSE)
```

Arguments

x	An object of the class "solist", essentially a list of two-dimensional spatial datasets.
...	Arguments passed to plot when generating each plot panel.
main	Overall heading for the plot.
arrange	Logical flag indicating whether to plot the objects side-by-side on a single page (arrange=TRUE) or plot them individually in a succession of frames (arrange=FALSE).
nrows,ncols	Optional. The number of rows/columns in the plot layout (assuming arrange=TRUE). You can specify either or both of these numbers.

`main.panel`	Optional. A character string, or a vector of character strings, giving the headings for each of the objects.
`mar.panel`	Size of the margins outside each plot panel. A numeric vector of length 4 giving the bottom, left, top, and right margins in that order. (Alternatively the vector may have length 1 or 2 and will be replicated to length 4). See the section on *Spacing between plots*.
`hsep,vsep`	Additional horizontal and vertical separation between plot panels, expressed in the same units as `mar.panel`.
`panel.begin,panel.end`	Optional. Functions that will be executed before and after each panel is plotted. See Details.
`panel.args`	Optional. Function that determines different plot arguments for different panels. See Details.
`panel.begin.args`	Optional. List of additional arguments for `panel.begin` when it is a function.
`panel.end.args`	Optional. List of additional arguments for `panel.end` when it is a function.
`panel.vpad`	Amount of extra vertical space that should be allowed for the title of each panel, if a title will be displayed. Expressed as a fraction of the height of the panel. Applies only when `equal.scales=FALSE` (the default).
`plotcommand`	Optional. Character string containing the name of the command that should be executed to plot each panel.
`adorn.left,adorn.right,adorn.top,adorn.bottom`	Optional. Functions (with no arguments) that will be executed to generate additional plots at the margins (left, right, top and/or bottom, respectively) of the array of plots.
`adorn.size`	Relative width (as a fraction of the other panels' widths) of the margin plots.
`equal.scales`	Logical value indicating whether the components should be plotted at (approximately) the same physical scale.
`halign,valign`	Logical values indicating whether panels in a column should be aligned to the same x coordinate system (`halign=TRUE`) and whether panels in a row should be aligned to the same y coordinate system (`valign=TRUE`). These are applicable only if `equal.scales=TRUE`.

Details

This is the `plot` method for the class `"solist"`.

An object of class `"solist"` represents a list of two-dimensional spatial datasets. This is the `plot` method for such objects.

In the **spatstat** package, various functions produce an object of class `"solist"`. These objects can be plotted in a nice arrangement using `plot.solist`. See the Examples.

The argument `panel.args` determines extra graphics parameters for each panel. It should be a function that will be called as `panel.args(i)` where `i` is the panel number. Its return value should be a list of graphics parameters that can be passed to the relevant `plot` method. These parameters override any parameters specified in the `...` arguments.

The arguments `panel.begin` and `panel.end` determine graphics that will be plotted before and after each panel is plotted. They may be objects of some class that can be plotted with the generic `plot` command. Alternatively they may be functions that will be called as `panel.begin(i, y, main=main.panel[i])` and `panel.end(i, y, add=TRUE)` where `i` is the panel number and `y = x[[i]]`.

If all entries of x are pixel images, the function image.listof is called to control the plotting. The arguments equal.ribbon and col can be used to determine the colour map or maps applied.

If equal.scales=FALSE (the default), then the plot panels will have equal height on the plot device (unless there is only one column of panels, in which case they will have equal width on the plot device). This means that the objects are plotted at different physical scales, by default.

If equal.scales=TRUE, then the dimensions of the plot panels on the plot device will be proportional to the spatial dimensions of the corresponding components of x. This means that the objects will be plotted at *approximately* equal physical scales. If these objects have very different spatial sizes, the plot command could fail (when it tries to plot the smaller objects at a tiny scale), with an error message that the figure margins are too large.

The objects will be plotted at *exactly* equal physical scales, and *exactly* aligned on the device, under the following conditions:

- every component of x is a spatial object whose position can be shifted by shift;
- panel.begin and panel.end are either NULL or they are spatial objects whose position can be shifted by shift;
- adorn.left, adorn.right, adorn.top and adorn.bottom are all NULL.

Another special case is when every component of x is an object of class "fv" representing a function. If equal.scales=TRUE then all these functions will be plotted with the same axis scales (i.e. with the same xlim and the same ylim).

Value

Null.

Spacing between plots

The spacing between individual plots is controlled by the parameters mar.panel, hsep and vsep.

If equal.scales=FALSE, the plot panels are logically separate plots. The margins for each panel are determined by the argument mar.panel which becomes the graphics parameter mar described in the help file for par. One unit of mar corresponds to one line of text in the margin. If hsep or vsep are present, mar.panel is augmented by c(vsep, hsep, vsep, hsep)/2.

If equal.scales=TRUE, all the plot panels are drawn in the same coordinate system which represents a physical scale. The unit of measurement for mar.panel[1,3] is one-sixth of the greatest height of any object plotted in the same row of panels, and the unit for mar.panel[2,4] is one-sixth of the greatest width of any object plotted in the same column of panels. If hsep or vsep are present, they are interpreted in the same units as mar.panel[2] and mar.panel[1] respectively.

Error messages

If the error message 'Figure margins too large' occurs, this generally means that one of the objects had a much smaller physical scale than the others. Ensure that equal.scales=FALSE and increase the values of mar.panel.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

plot.anylist, contour.listof, image.listof, density.splitppp

Examples

```
# Intensity estimate of multitype point pattern
 plot(D <- density(split(amacrine)))
 plot(D, main="", equal.ribbon=TRUE,
      panel.end=function(i,y,...){contour(y, ...)})
```

---

plot.splitppp *Plot a List of Point Patterns*

Description

Plots a list of point patterns.

Usage

```
  ## S3 method for class 'splitppp'
plot(x, ..., main)
```

Arguments

x	A named list of point patterns, typically obtained from split.ppp.
...	Arguments passed to plot.listof which control the layout of the plot panels, their appearance, and the plot behaviour in individual plot panels.
main	Optional main title for the plot.

Details

This is the plot method for the class "splitppp". It is typically used to plot the result of the function split.ppp.

The argument x should be a named list of point patterns (objects of class "ppp", see ppp.object). Each of these point patterns will be plotted in turn using plot.ppp.

Plotting is performed by plot.listof.

Value

Null.

Error messages

If the error message 'Figure margins too large' occurs, ensure that equal.scales=FALSE and increase the values of mar.panel.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

plot.listof for arguments controlling the plot.

split.ppp, plot.ppp, ppp.object.

Examples

```
# Multitype point pattern
 plot(split(amacrine))
 plot(split(amacrine), main="",
     panel.begin=function(i, y, ...) { plot(density(y), ribbon=FALSE, ...) })
```

---

plot.ssf *Plot a Spatially Sampled Function*

Description

Plot a spatially sampled function object.

Usage

```
## S3 method for class 'ssf'
plot(x, ...,
                    how = c("smoothed", "nearest", "points"),
                    style = c("image", "contour", "imagecontour"),
                    sigma = NULL, contourargs=list())

## S3 method for class 'ssf'
image(x, ...)

## S3 method for class 'ssf'
contour(x, ..., main, sigma = NULL)
```

Arguments

x	Spatially sampled function (object of class "ssf").
...	Arguments passed to image.default or plot.ppp to control the plot.
how	Character string determining whether to display the function values at the data points (how="points"), a smoothed interpolation of the function (how="smoothed"), or the function value at the nearest data point (how="nearest").
style	Character string indicating whether to plot the smoothed function as a colour image, a contour map, or both.
contourargs	Arguments passed to contour.default to control the contours, if style="contour" or style="imagecontour".
sigma	Smoothing bandwidth for smooth interpolation.
main	Optional main title for the plot.

Details

These are methods for the generic plot, image and contour for the class "ssf".

An object of class "ssf" represents a function (real- or vector-valued) that has been sampled at a finite set of points.

For plot.ssf there are three types of display. If how="points" the exact function values will be displayed as circles centred at the locations where they were computed. If how="smoothed" (the default) these values will be kernel-smoothed using Smooth.ppp and displayed as a pixel image. If how="nearest" the values will be interpolated by nearest neighbour interpolation using nnmark and displayed as a pixel image.

For image.ssf and contour.ssf the values are kernel-smoothed before being displayed.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Baddeley, A. (2016) Local composite likelihood for spatial point processes. *Spatial Statistics*, in press.

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

See Also

ssf

Examples

```
a <- ssf(cells, nndist(cells, k=1:3))
plot(a, how="points")
plot(a, how="smoothed")
plot(a, how="nearest")
```

---

plot.symbolmap *Plot a Graphics Symbol Map*

Description

Plot a representation of a graphics symbol map, similar to a plot legend.

Usage

```
## S3 method for class 'symbolmap'
plot(x, ..., main, xlim = NULL, ylim = NULL,
                         vertical = FALSE,
                         side = c("bottom", "left", "top", "right"),
                         annotate = TRUE, labelmap = NULL, add = FALSE,
                         nsymbols = NULL)
```

Arguments

`x`	Graphics symbol map (object of class "symbolmap").
`...`	Additional graphics arguments passed to points, symbols or axis.
`main`	Main title for the plot. A character string.
`xlim,ylim`	Coordinate limits for the plot. Numeric vectors of length 2.
`vertical`	Logical. Whether to plot the symbol map in a vertical orientation.
`side`	Character string specifying the position of the text that annotates the symbols.
`annotate`	Logical. Whether to annotate the symbols with labels.
`labelmap`	Transformation of the labels. A function or a scale factor which will be applied to the data values corresponding to the plotted symbols.
`add`	Logical value indicating whether to add the plot to the current plot (add=TRUE) or to initialise a new plot.
`nsymbols`	Optional. The number of symbols that should be displayed. (This may not be exactly obeyed.)

Details

A graphics symbol map is an association between data values and graphical symbols.

This command plots the graphics symbol map itself, in the style of a plot legend.

Value

None.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

symbolmap to create a symbol map.

invoke.symbolmap to apply the symbol map to some data and plot the resulting symbols.

Examples

```
g <- symbolmap(inputs=letters[1:10], pch=11:20)
plot(g)

g2 <- symbolmap(range=c(-1,1),
                  shape=function(x) ifelse(x > 0, "circles", "squares"),
                  size=function(x) sqrt(ifelse(x > 0, x/pi, -x)),
                  bg = function(x) ifelse(abs(x) < 1, "red", "black"))
plot(g2, vertical=TRUE, side="left", col.axis="blue", cex.axis=2)
```

plot.tess	*Plot a tessellation*

Description

Plots a tessellation.

Usage

```
## S3 method for class 'tess'
plot(x, ..., main, add=FALSE,
                    show.all=!add, col=NULL, do.plot=TRUE,
                    do.labels=FALSE,
                    labels=tilenames(x), labelargs=list())
```

Arguments

`x`	Tessellation (object of class `"tess"`) to be plotted.
`...`	Arguments controlling the appearance of the plot.
`main`	Heading for the plot. A character string.
`add`	Logical. Determines whether the tessellation plot is added to the existing plot.
`show.all`	Logical value indicating whether to plot everything including the main title and the observation window of `x`.
`col`	Colour of the tile boundaries. A character string. Ignored for pixel tessellations.
`do.plot`	Logical value indicating whether to actually perform the plot.
`do.labels`	Logical value indicating whether to show a text label for each tile of the tessellation.
`labels`	Character vector of labels for the tiles.
`labelargs`	List of arguments passed to `text.default` to control display of the text labels.

Details

This is a method for the generic `plot` function for the class `"tess"` of tessellations (see `tess`).

The arguments `...` control the appearance of the plot. They are passed to `segments`, `plot.owin` or `plot.im`, depending on the type of tessellation.

Value

(Invisible) window of class `"owin"` specifying a bounding box for the plot (including a colour ribbon if plotted).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`tess`

Examples

```
A <- tess(xgrid=0:4,ygrid=0:4)
plot(A, col="blue", lwd=2, lty=2)
B <- A[c(1, 2, 5, 7, 9)]
plot(B, hatch=TRUE)
v <- as.im(function(x,y){factor(round(5 * (x^2 + y^2)))}, W=owin())
levels(v) <- letters[seq(length(levels(v)))]
E <- tess(image=v)
plot(E)
```

`plot.textstring` *Plot a Text String*

Description

Plots an object of class "textstring".

Usage

```
## S3 method for class 'textstring'
plot(x, ..., do.plot = TRUE)
```

Arguments

x	Object of class "textstring" to be plotted. This object is created by the command textstring.
...	Additional graphics arguments passed to text to control the plotting of text.
do.plot	Logical value indicating whether to actually plot the text.

Details

The argument x should be an object of class "textstring" created by the command textstring.

This function displays the text using text.

Value

A window (class "owin") enclosing the plotted graphics.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

onearrow, yardstick

Examples

```
W <- Window(humberside)
te <- textstring(centroid.owin(W), txt="Humberside", cex=2.5)
plot(layered(W, te), main="")
```

plot.texturemap *Plot a Texture Map*

Description

Plot a representation of a texture map, similar to a plot legend.

Usage

```
## S3 method for class 'texturemap'
plot(x, ..., main, xlim = NULL, ylim = NULL,
                             vertical = FALSE, axis = TRUE,
                             labelmap = NULL, gap = 0.25,
                             spacing = NULL, add = FALSE)
```

Arguments

x	Texture map object (class "texturemap").
...	Additional graphics arguments passed to add.texture or axis.default.
main	Main title for plot.
xlim,ylim	Optional vectors of length 2 giving the x and y limits of the plot.
vertical	Logical value indicating whether to arrange the texture boxes in a vertical column (vertical=TRUE or a horizontal row (vertical=FALSE, the default).
axis	Logical value indicating whether to plot an axis line joining the texture boxes.
labelmap	Optional. A function which will be applied to the data values (the inputs of the texture map) before they are displayed on the plot.
gap	Separation between texture boxes, as a fraction of the width or height of a box.
spacing	Argument passed to add.texture controlling the density of lines in a texture. Expressed in spatial coordinate units.
add	Logical value indicating whether to add the graphics to an existing plot (add=TRUE) or to initialise a new plot (add=FALSE, the default).

Details

A texture map is an association between data values and graphical textures. An object of class "texturemap" represents a texture map. Such objects are returned from the plotting function textureplot, and can be created directly by the function texturemap.

This function plot.texturemap is a method for the generic plot for the class "texturemap". It displays a sample of each of the textures in the texture map, in a separate box, annotated by the data value which is mapped to that texture.

The arrangement and position of the boxes is controlled by the arguments vertical, xlim, ylim and gap.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
Rolf Turner <r.turner@auckland.ac.nz>
and Ege Rubak <rubak@math.aau.dk>

See Also

texturemap, textureplot, add.texture.

Examples

```
tm <- texturemap(c("First", "Second", "Third"), 2:4, col=2:4)
plot(tm, vertical=FALSE)
## abbreviate the labels
plot(tm, labelmap=function(x) substr(x, 1, 2))
```

---

plot.yardstick *Plot a Yardstick or Scale Bar*

Description

Plots an object of class "yardstick".

Usage

```
## S3 method for class 'yardstick'
plot(x, ...,
        angle = 20, frac = 1/8,
        split = FALSE, shrink = 1/4,
        pos = NULL,
        txt.args=list(),
        txt.shift=c(0,0),
        do.plot = TRUE)
```

Arguments

x	Object of class "yardstick" to be plotted. This object is created by the command yardstick.
...	Additional graphics arguments passed to segments to control the appearance of the line.
angle	Angle between the arrows and the line segment, in degrees.
frac	Length of arrow as a fraction of total length of the line segment.
split	Logical. If TRUE, then the line will be broken in the middle, and the text will be placed in this gap. If FALSE, the line will be unbroken, and the text will be placed beside the line.
shrink	Fraction of total length to be removed from the middle of the line segment, if split=TRUE.
pos	Integer (passed to text) determining the position of the annotation text relative to the line segment, if split=FALSE. Values of 1, 2, 3 and 4 indicate positions below, to the left of, above and to the right of the line, respectively.

txt.args	Optional list of additional arguments passed to text controlling the appearance of the text. Examples include adj, srt, col, cex, font.
txt.shift	Optional numeric vector of length 2 specifying displacement of the text position relative to the centre of the yardstick.
do.plot	Logical. Whether to actually perform the plot (do.plot=TRUE).

Details

A yardstick or scale bar is a line segment, drawn on any spatial graphics display, indicating the scale of the plot.

The argument x should be an object of class "yardstick" created by the command yardstick.

Value

A window (class "owin") enclosing the plotted graphics.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

yardstick

Examples

```
plot(owin(), main="Yardsticks")
ys <- yardstick(as.psp(list(xmid=0.5, ymid=0.1, length=0.4, angle=0),
                       window=owin(c(0.2, 0.8), c(0, 0.2))),
                txt="1 km")
plot(ys)
ys <- shift(ys, c(0, 0.3))
plot(ys, angle=90, frac=0.08)
ys <- shift(ys, c(0, 0.3))
plot(ys, split=TRUE)
```

---

points.lpp *Draw Points on Existing Plot*

Description

For a point pattern on a linear network, this function draws the coordinates of the points only, on the existing plot display.

Usage

```
## S3 method for class 'lpp'
points(x, ...)
```

Arguments

`x`	A point pattern on a linear network (object of class `"lpp"`).
`...`	Additional arguments passed to `points.default`.

Details

This is a method for the generic function `points` for the class `"lpp"` of point patterns on a linear network.

If `x` is a point pattern on a linear network, then `points(x)` plots the spatial coordinates of the points only, on the existing plot display, without plotting the underlying network. It is an error to call this function if a plot has not yet been initialised.

The spatial coordinates are extracted and passed to `points.default` along with any extra arguments. Arguments controlling the colours and the plot symbols are interpreted by `points.default`. For example, if the argument `col` is a vector, then the `i`th point is drawn in the colour `col[i]`.

Value

Null.

Difference from plot method

The more usual way to plot the points is using `plot.lpp`. For example `plot(x)` would plot both the points and the underlying network, while `plot(x, add=TRUE)` would plot only the points. The interpretation of arguments controlling the colours and plot symbols is different here: they determine a symbol map, as explained in the help for `plot.ppp`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`plot.lpp`, `points.default`

Examples

```
plot(Frame(spiders), main="Spiders on a Brick Wall")
points(spiders)
```

`pointsOnLines` *Place Points Evenly Along Specified Lines*

Description

Given a line segment pattern, place a series of points at equal distances along each line segment.

Usage

```
pointsOnLines(X, eps = NULL, np = 1000, shortok=TRUE)
```

Arguments

X	A line segment pattern (object of class "psp").
eps	Spacing between successive points.
np	Approximate total number of points (incompatible with eps).
shortok	Logical. If FALSE, very short segments (of length shorter than eps) will not generate any points. If TRUE, a very short segment will be represented by its midpoint.

Details

For each line segment in the pattern X, a succession of points is placed along the line segment. These points are equally spaced at a distance eps, except for the first and last points in the sequence.

The spacing eps is measured in coordinate units of X.

If eps is not given, then it is determined by eps = len/np where len is the total length of the segments in X. The actual number of points will then be slightly larger than np.

Value

A point pattern (object of class "ppp") in the same window as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

psp, ppp, runifpointOnLines

Examples

```
X <- psp(runif(20), runif(20), runif(20), runif(20),  window=owin())
Y <- pointsOnLines(X, eps=0.05)
plot(X, main="")
plot(Y, add=TRUE, pch="+")
```

Poisson *Poisson Point Process Model*

Description

Creates an instance of the Poisson point process model which can then be fitted to point pattern data.

Usage

```
Poisson()
```

Details

The function ppm, which fits point process models to point pattern data, requires an argument `interaction` of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Poisson process is provided by the value of the function `Poisson`.

This works for all types of Poisson processes including multitype and nonstationary Poisson processes.

Value

An object of class `"interact"` describing the interpoint interaction structure of the Poisson point process (namely, there are no interactions).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

ppm, Strauss

Examples

```
ppm(nztrees ~1, Poisson())
# fit the stationary Poisson process to 'nztrees'
# no edge correction needed

lon <- longleaf

longadult <- unmark(subset(lon, marks >= 30))
ppm(longadult ~ x, Poisson())
# fit the nonstationary Poisson process
# with intensity lambda(x,y) = exp( a + bx)

# trees marked by species
lans <- lansing

ppm(lans ~ marks, Poisson())
# fit stationary marked Poisson process
# with different intensity for each species

## Not run:
ppm(lansing ~ marks * polynom(x,y,3), Poisson())

## End(Not run)
# fit nonstationary marked Poisson process
# with different log-cubic trend for each species
```

polynom *Polynomial in One or Two Variables*

Description

This function is used to represent a polynomial term in a model formula. It computes the homogeneous terms in the polynomial of degree `n` in one variable `x` or two variables `x,y`.

Usage

```
polynom(x, ...)
```

Arguments

`x`	A numerical vector.
`...`	Either a single integer `n` specifying the degree of the polynomial, or two arguments `y,n` giving another vector of data `y` and the degree of the polynomial.

Details

This function is typically used inside a model formula in order to specify the most general possible polynomial of order `n` involving one numerical variable `x` or two numerical variables `x,y`.

It is equivalent to `poly(, raw=TRUE)`.

If only one numerical vector argument `x` is given, the function computes the vectors `x^k` for `k = 1, 2, ..., n`. These vectors are combined into a matrix with `n` columns.

If two numerical vector arguments `x,y` are given, the function computes the vectors `x^k * y^m` for `k >= 0` and `m >= 0` satisfying `0 < k + m <= n`. These vectors are combined into a matrix with one column for each homogeneous term.

Value

A numeric matrix, with rows corresponding to the entries of `x`, and columns corresponding to the terms in the polynomial.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`poly`, `harmonic`

Examples

```
x <- 1:4
y <- 10 * (0:3)
polynom(x, 3)
polynom(x, y, 3)
```

pool — *Pool Data*

Description

Pool the data from several objects of the same class.

Usage

```
pool(...)
```

Arguments

...	Objects of the same type.

Details

The function pool is generic. There are methods for several classes, listed below.

pool is used to combine the data from several objects of the same type, and to compute statistics based on the combined dataset. It may be used to pool the estimates obtained from replicated datasets. It may also be used in high-performance computing applications, when the objects ... have been computed on different processors or in different batch runs, and we wish to combine them.

Value

An object of the same class as the arguments ....

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

pool.envelope, pool.fasp, pool.rat, pool.fv

pool.anylist — *Pool Data from a List of Objects*

Description

Pool the data from the objects in a list.

Usage

```
## S3 method for class 'anylist'
pool(x, ...)
```

Arguments

`x`	A list, belonging to the class `"anylist"`, containing objects that can be pooled.
`...`	Optional additional objects which can be pooled with the elements of `x`.

Details

The function `pool` is generic. Its purpose is to combine data from several objects of the same type (typically computed from different datasets) into a common, pooled estimate.

The function `pool.anyist` is the method for the class `"anylist"`. It is used when the objects to be pooled are given in a list `x`.

Each of the elements of the list `x`, and each of the subsequent arguments `...` if provided, must be an object of the same class.

Value

An object of the same class as each of the entries in `x`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`anylist`, `pool`.

Examples

```
Keach <- anylapply(waterstriders, Kest, ratio=TRUE, correction="iso")
K <- pool(Keach)
```

`pool.envelope` *Pool Data from Several Envelopes*

Description

Pool the simulation data from several simulation envelopes (objects of class `"envelope"`) and compute a new envelope.

Usage

```
## S3 method for class 'envelope'
pool(..., savefuns=FALSE, savepatterns=FALSE)
```

Arguments

`...`	Objects of class `"envelope"`.
`savefuns`	Logical flag indicating whether to save all the simulated function values.
`savepatterns`	Logical flag indicating whether to save all the simulated point patterns.

Details

The function pool is generic. This is the method for the class "envelope" of simulation envelopes. It is used to combine the simulation data from several simulation envelopes and to compute an envelope based on the combined data.

Each of the arguments ... must be an object of class "envelope". These envelopes must be compatible, in that they are envelopes for the same function, and were computed using the same options.

- In normal use, each envelope object will have been created by running the command envelope with the argument savefuns=TRUE. This ensures that each object contains the simulated data (summary function values for the simulated point patterns) that were used to construct the envelope.

  The simulated data are extracted from each object and combined. A new envelope is computed from the combined set of simulations.
- Alternatively, if each envelope object was created by running envelope with VARIANCE=TRUE, then the saved functions are not required.

  The sample means and sample variances from each envelope will be pooled. A new envelope is computed from the pooled mean and variance.

Warnings or errors will be issued if the envelope objects ... appear to be incompatible. Apart from these basic checks, the code is not smart enough to decide whether it is sensible to pool the data.

To modify the envelope parameters or the type of envelope that is computed, first pool the envelope data using pool.envelope, then use envelope.envelope to modify the envelope parameters.

Value

An object of class "envelope".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

envelope, envelope.envelope, pool, pool.fasp

Examples

```
E1 <- envelope(cells, Kest, nsim=10, savefuns=TRUE)
E2 <- envelope(cells, Kest, nsim=20, savefuns=TRUE)
pool(E1, E2)

V1 <- envelope(E1, VARIANCE=TRUE)
V2 <- envelope(E2, VARIANCE=TRUE)
pool(V1, V2)
```

pool.fasp *Pool Data from Several Function Arrays*

Description

Pool the simulation data from several function arrays (objects of class "fasp") and compute a new function array.

Usage

```
## S3 method for class 'fasp'
pool(...)
```

Arguments

`...` Objects of class "fasp".

Details

The function `pool` is generic. This is the method for the class "fasp" of function arrays. It is used to combine the simulation data from several arrays of simulation envelopes and to compute a new array of envelopes based on the combined data.

Each of the arguments `...` must be a function array (object of class "fasp") containing simulation envelopes. This is typically created by running the command `alltypes` with the arguments `envelope=TRUE` and `savefuns=TRUE`. This ensures that each object is an array of simulation envelopes, and that each envelope contains the simulated data (summary function values) that were used to construct the envelope.

The simulated data are extracted from each object and combined. A new array of envelopes is computed from the combined set of simulations.

Warnings or errors will be issued if the objects `...` appear to be incompatible. However, the code is not smart enough to decide whether it is sensible to pool the data.

Value

An object of class "fasp".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`fasp`, `alltypes`, `pool.envelope`, `pool`

Examples

```
data(amacrine)
A1 <- alltypes(amacrine,"K",nsim=9,envelope=TRUE,savefuns=TRUE)
A2 <- alltypes(amacrine,"K",nsim=10,envelope=TRUE,savefuns=TRUE)
pool(A1, A2)
```

pool.fv *Pool Several Functions*

Description

Combine several summary functions into a single function.

Usage

```
## S3 method for class 'fv'
pool(..., weights=NULL, relabel=TRUE, variance=TRUE)
```

Arguments

...	Objects of class "fv".
weights	Optional numeric vector of weights for the functions.
relabel	Logical value indicating whether the columns of the resulting function should be labelled to show that they were obtained by pooling.
variance	Logical value indicating whether to compute the sample variance and related terms.

Details

The function pool is generic. This is the method for the class "fv" of summary functions. It is used to combine several estimates of the same function into a single function.

Each of the arguments ... must be an object of class "fv". They must be compatible, in that they are estimates of the same function, and were computed using the same options.

The sample mean and sample variance of the corresponding estimates will be computed.

Value

An object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

pool, pool.anylist, pool.rat

Examples

```
K <- lapply(waterstriders, Kest, correction="iso")
Kall <- pool(K[[1]], K[[2]], K[[3]])
Kall <- pool(as.anylist(K))
plot(Kall, cbind(pooliso, pooltheo) ~ r,
           shade=c("loiso", "hiiso"),
           main="Pooled K function of waterstriders")
```

`pool.quadrattest` *Pool Several Quadrat Tests*

Description

Pool several quadrat tests into a single quadrat test.

Usage

```
## S3 method for class 'quadrattest'
pool(..., df=NULL, df.est=NULL, nsim=1999,
                                 Xname=NULL, CR=NULL)
```

Arguments

`...`	Any number of objects, each of which is a quadrat test (object of class `"quadrattest"`).
`df`	Optional. Number of degrees of freedom of the test statistic. Relevant only for χ^2 tests. Incompatible with `df.est`.
`df.est`	Optional. The number of fitted parameters, or the number of degrees of freedom lost by estimation of parameters. Relevant only for χ^2 tests. Incompatible with `df`.
`nsim`	Number of simulations, for Monte Carlo test.
`Xname`	Optional. Name of the original data.
`CR`	Optional. Numeric value of the Cressie-Read exponent `CR` overriding the value used in the tests.

Details

The function `pool` is generic. This is the method for the class `"quadrattest"`.

An object of class `"quadrattest"` represents a χ^2 test or Monte Carlo test of goodness-of-fit for a point process model, based on quadrat counts. Such objects are created by the command `quadrat.test`.

Each of the arguments `...` must be an object of class `"quadrattest"`. They must all be the same type of test (chi-squared test or Monte Carlo test, conditional or unconditional) and must all have the same type of alternative hypothesis.

The test statistic of the pooled test is the Pearson X^2 statistic taken over all cells (quadrats) of all tests. The p value of the pooled test is then computed using either a Monte Carlo test or a χ^2 test.

For a pooled χ^2 test, the number of degrees of freedom of the combined test is computed by adding the degrees of freedom of all the tests (equivalent to assuming the tests are independent) unless it is determined by the arguments `df` or `df.est`. The resulting p value is computed to obtain the pooled test.

For a pooled Monte Carlo test, new simulations are performed to determine the pooled Monte Carlo p value.

Value

Another object of class `"quadrattest"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

pool, quadrat.test

Examples

```
Y <- split(humberside)
test1 <- quadrat.test(Y[[1]])
test2 <- quadrat.test(Y[[2]])
pool(test1, test2, Xname="Humberside")
```

---

pool.rat *Pool Data from Several Ratio Objects*

Description

Pool the data from several ratio objects (objects of class "rat") and compute a pooled estimate.

Usage

```
## S3 method for class 'rat'
pool(..., weights=NULL, relabel=TRUE, variance=TRUE)
```

Arguments

...	Objects of class "rat".
weights	Numeric vector of weights.
relabel	Logical value indicating whether the result should be relabelled to show that it was obtained by pooling.
variance	Logical value indicating whether to compute the sample variance and related terms.

Details

The function pool is generic. This is the method for the class "rat" of ratio objects. It is used to combine several estimates of the same quantity when each estimate is a ratio.

Each of the arguments ... must be an object of class "rat" representing a ratio object (basically a numerator and a denominator; see rat). We assume that these ratios are all estimates of the same quantity.

If the objects are called $R_1, \ldots, R_n$ and if R_i has numerator Y_i and denominator X_i, so that notionally $R_i = Y_i/X_i$, then the pooled estimate is the ratio-of-sums estimator

$$R = \frac{\sum_i Y_i}{\sum_i X_i}.$$

The standard error of R is computed using the delta method as described in Baddeley *et al.* (1993) or Cochran (1977, pp 154, 161).

If the argument `weights` is given, it should be a numeric vector of length equal to the number of objects to be pooled. The pooled estimator is the ratio-of-sums estimator

$$R = \frac{\sum_i w_i Y_i}{\sum_i w_i X_i}$$

where `w_iw[i]` is the `i`th weight.

This calculation is implemented only for certain classes of objects where the arithmetic can be performed.

This calculation is currently implemented only for objects which also belong to the class `"fv"` (function value tables). For example, if `Kest` is called with argument `ratio=TRUE`, the result is a suitable object (belonging to the classes `"rat"` and `"fv"`).

Warnings or errors will be issued if the ratio objects `...` appear to be incompatible. However, the code is not smart enough to decide whether it is sensible to pool the data.

Value

An object of the same class as the input.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>, Rolf Turner <`r.turner@auckland.ac.nz`> and Ege Rubak <`rubak@math.aau.dk`>.

References

Baddeley, A.J, Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.

Cochran, W.G. (1977) *Sampling techniques*, 3rd edition. New York: John Wiley and Sons.

See Also

`rat`, `pool`, `pool.fv`, `Kest`

Examples

```
K1 <- Kest(runifpoint(42), ratio=TRUE, correction="iso")
K2 <- Kest(runifpoint(42), ratio=TRUE, correction="iso")
K3 <- Kest(runifpoint(42), ratio=TRUE, correction="iso")
K <- pool(K1, K2, K3)
plot(K, pooliso ~ r, shade=c("hiiso", "loiso"))
```

---

pp3 *Three Dimensional Point Pattern*

Description

Create a three-dimensional point pattern

Usage

```
pp3(x, y, z, ...)
```

Arguments

`x,y,z`	Numeric vectors of equal length, containing Cartesian coordinates of points in three-dimensional space.
`...`	Arguments passed to `as.box3` to determine the three-dimensional box in which the points have been observed.

Details

An object of class "pp3" represents a pattern of points in three-dimensional space. The points are assumed to have been observed by exhaustively inspecting a three-dimensional rectangular box. The boundaries of the box are included as part of the dataset.

Value

Object of class "pp3" representing a three dimensional point pattern. Also belongs to class "ppx".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`box3`, `print.pp3`, `ppx`

Examples

```
X <- pp3(runif(10), runif(10), runif(10), box3(c(0,1)))
```

---

ppm *Fit Point Process Model to Data*

Description

Fits a point process model to an observed point pattern.

Usage

```
ppm(Q, ...)

   ## S3 method for class 'formula'
ppm(Q, interaction=NULL, ..., data=NULL, subset)
```

Arguments

`Q`	A `formula` in the R language describing the model to be fitted.
`interaction`	An object of class "interact" describing the point process interaction structure, or a function that makes such an object, or `NULL` indicating that a Poisson process (stationary or nonstationary) should be fitted.
`...`	Arguments passed to `ppm.ppp` or `ppm.quad` to control the model-fitting process.

data	Optional. The values of spatial covariates (other than the Cartesian coordinates) required by the model. Either a data frame, or a list whose entries are images, functions, windows, tessellations or single numbers. See Details.
subset	Optional. An expression (which may involve the names of the Cartesian coordinates x and y and the names of entries in data) defining a subset of the spatial domain, to which the model-fitting should be restricted. The result of evaluating the expression should be either a logical vector, or a window (object of class "owin") or a logical-valued pixel image (object of class "im").

Details

This function fits a point process model to an observed point pattern. The model may include spatial trend, interpoint interaction, and dependence on covariates.

The model fitted by ppm is either a Poisson point process (in which different points do not interact with each other) or a Gibbs point process (in which different points typically inhibit each other). For clustered point process models, use kppm.

The function ppm is generic, with methods for the classes formula, ppp and quad. This page describes the method for a formula.

The first argument is a formula in the R language describing the spatial trend model to be fitted. It has the general form pattern ~ trend where the left hand side pattern is usually the name of a spatial point pattern (object of class "ppp") to which the model should be fitted, or an expression which evaluates to a point pattern; and the right hand side trend is an expression specifying the spatial trend of the model.

Systematic effects (spatial trend and/or dependence on spatial covariates) are specified by the trend expression on the right hand side of the formula. The trend may involve the Cartesian coordinates x, y, the marks marks, the names of entries in the argument data (if supplied), or the names of objects that exist in the R session. The trend formula specifies the **logarithm** of the intensity of a Poisson process, or in general, the logarithm of the first order potential of the Gibbs process. The formula should not use any names beginning with .mpl as these are reserved for internal use. If the formula is pattern~1, then the model to be fitted is stationary (or at least, its first order potential is constant).

The symbol . in the trend expression stands for all the covariates supplied in the argument data. For example the formula pattern ~ . indicates an additive model with a main effect for each covariate in data.

Stochastic interactions between random points of the point process are defined by the argument interaction. This is an object of class "interact" which is initialised in a very similar way to the usage of family objects in glm and gam. The interaction models currently available are: AreaInter, BadGey, Concom, DiggleGatesStibbard, DiggleGratton, Fiksel, Geyer, Hardcore, Hybrid, LennardJones, MultiStrauss, MultiStraussHard, OrdThresh, Ord, Pairwise, PairPiece, Penttinen, Poisson, Saturated, SatPiece, Softcore, Strauss, StraussHard and Triplets. See the examples below. Note that it is possible to combine several interactions using Hybrid.

If interaction is missing or NULL, then the model to be fitted has no interpoint interactions, that is, it is a Poisson process (stationary or nonstationary according to trend). In this case the methods of maximum pseudolikelihood and maximum logistic likelihood coincide with maximum likelihood.

The fitted point process model returned by this function can be printed (by the print method print.ppm) to inspect the fitted parameter values. If a nonparametric spatial trend was fitted, this can be extracted using the predict method predict.ppm.

To fit a model involving spatial covariates other than the Cartesian coordinates x and y, the values of the covariates should either be supplied in the argument data, or should be stored in objects that exist in the R session. Note that it is not sufficient to have observed the covariate only at the points of the data point pattern; the covariate must also have been observed at other locations in the window.

If it is given, the argument `data` is typically a list, with names corresponding to variables in the `trend` formula. Each entry in the list is either

a pixel image, giving the values of a spatial covariate at a fine grid of locations. It should be an object of class `"im"`, see `im.object`.

a function, which can be evaluated at any location `(x,y)` to obtain the value of the spatial covariate. It should be a `function(x, y)` or `function(x, y, ...)` in the R language. The first two arguments of the function should be the Cartesian coordinates x and y. The function may have additional arguments; if the function does not have default values for these additional arguments, then the user must supply values for them, in `covfunargs`. See the Examples.

a window, interpreted as a logical variable which is `TRUE` inside the window and `FALSE` outside it. This should be an object of class `"owin"`.

a tessellation, interpreted as a factor covariate. For each spatial location, the factor value indicates which tile of the tessellation it belongs to. This should be an object of class `"tess"`.

a single number, indicating a covariate that is constant in this dataset.

The software will look up the values of each covariate at the required locations (quadrature points).

Note that, for covariate functions, only the *name* of the function appears in the trend formula. A covariate function is treated as if it were a single variable. The function arguments do not appear in the trend formula. See the Examples.

If `data` is a list, the list entries should have names corresponding to (some of) the names of covariates in the model formula `trend`. The variable names `x`, `y` and `marks` are reserved for the Cartesian coordinates and the mark values, and these should not be used for variables in `data`.

Alternatively, `data` may be a data frame giving the values of the covariates at specified locations. Then `pattern` should be a quadrature scheme (object of class `"quad"`) giving the corresponding locations. See `ppm.quad` for details.

Value

An object of class `"ppm"` describing a fitted point process model.

See `ppm.object` for details of the format of this object and methods available for manipulating it.

Interaction parameters

Apart from the Poisson model, every point process model fitted by `ppm` has parameters that determine the strength and range of 'interaction' or dependence between points. These parameters are of two types:

regular parameters: A parameter ϕ is called *regular* if the log likelihood is a linear function of θ where $\theta = \theta(\psi)$ is some transformation of ψ. [Then θ is called the canonical parameter.]

irregular parameters Other parameters are called *irregular*.

Typically, regular parameters determine the 'strength' of the interaction, while irregular parameters determine the 'range' of the interaction. For example, the Strauss process has a regular parameter γ controlling the strength of interpoint inhibition, and an irregular parameter r determining the range of interaction.

The `ppm` command is only designed to estimate regular parameters of the interaction. It requires the values of any irregular parameters of the interaction to be fixed. For example, to fit a Strauss process model to the `cells` dataset, you could type `ppm(cells ~ 1, Strauss(r=0.07))`. Note that the value of the irregular parameter r must be given. The result of this command will be a fitted model in which the regular parameter γ has been estimated.

To determine the irregular parameters, there are several practical techniques, but no general statistical theory available. Useful techniques include maximum profile pseudolikelihood, which is implemented in the command profilepl, and Newton-Raphson maximisation, implemented in the experimental command ippm.

Some irregular parameters can be estimated directly from data: the hard-core radius in the model Hardcore and the matrix of hard-core radii in MultiHard can be estimated easily from data. In these cases, ppm allows the user to specify the interaction without giving the value of the irregular parameter. The user can give the hard core interaction as interaction=Hardcore() or even interaction=Hardcore, and the hard core radius will then be estimated from the data.

Technical Warnings and Error Messages

See ppm.ppp for some technical warnings about the weaknesses of the algorithm, and explanation of some common error messages.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A., Coeurjolly, J.-F., Rubak, E. and Waagepetersen, R. (2014) Logistic regression for spatial Gibbs point processes. *Biometrika* **101** (2) 377–392.

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** 283–322.

Berman, M. and Turner, T.R. (1992) Approximating point process likelihoods with GLIM. *Applied Statistics* **41**, 31–38.

Besag, J. (1975) Statistical analysis of non-lattice data. *The Statistician* **24**, 179-195.

Diggle, P.J., Fiksel, T., Grabarnik, P., Ogata, Y., Stoyan, D. and Tanemura, M. (1994) On parameter estimation for pairwise interaction processes. *International Statistical Review* **62**, 99-117.

Huang, F. and Ogata, Y. (1999) Improvements of the maximum pseudo-likelihood estimators in various spatial statistical models. *Journal of Computational and Graphical Statistics* **8**, 510–530.

Jensen, J.L. and Moeller, M. (1991) Pseudolikelihood for exponential family models of spatial point processes. *Annals of Applied Probability* **1**, 445–461.

Jensen, J.L. and Kuensch, H.R. (1994) On asymptotic normality of pseudo likelihood estimates for pairwise interaction processes, *Annals of the Institute of Statistical Mathematics* **46**, 475–486.

See Also

ppm.ppp and ppm.quad for more details on the fitting technique and edge correction.

ppm.object for details of how to print, plot and manipulate a fitted model.

ppp and quadscheme for constructing data.

Interactions: AreaInter, BadGey, Concom, DiggleGatesStibbard, DiggleGratton, Fiksel, Geyer, Hardcore, Hybrid, LennardJones, MultiStrauss, MultiStraussHard, OrdThresh, Ord, Pairwise, PairPiece, Penttinen, Poisson, Saturated, SatPiece, Softcore, Strauss, StraussHard and Triplets.

See `profilepl` for advice on fitting nuisance parameters in the interaction, and `ippm` for irregular parameters in the trend.

See `valid.ppm` and `project.ppm` for ensuring the fitted model is a valid point process.

See `kppm` for fitting Cox point process models and cluster point process models, and `dppm` for fitting determinantal point process models.

Examples

```
 # fit the stationary Poisson process
 # to point pattern 'nztrees'

 ppm(nztrees ~ 1)

 ## Not run:
 Q <- quadscheme(nztrees)
 ppm(Q ~ 1)
 # equivalent.

## End(Not run)

fit1 <- ppm(nztrees ~ x)
 # fit the nonstationary Poisson process
 # with intensity function lambda(x,y) = exp(a + bx)
 # where x,y are the Cartesian coordinates
 # and a,b are parameters to be estimated

fit1
coef(fit1)
coef(summary(fit1))

## Not run:
 ppm(nztrees ~ polynom(x,2))

## End(Not run)


 # fit the nonstationary Poisson process
 # with intensity function lambda(x,y) = exp(a + bx + cx^2)

 ## Not run:
 library(splines)
 ppm(nztrees ~ bs(x,df=3))

## End(Not run)
 #       WARNING: do not use predict.ppm() on this result
 # Fits the nonstationary Poisson process
 # with intensity function lambda(x,y) = exp(B(x))
 # where B is a B-spline with df = 3

## Not run:
 ppm(nztrees ~ 1, Strauss(r=10), rbord=10)

## End(Not run)

 # Fit the stationary Strauss process with interaction range r=10
 # using the border method with margin rbord=10
```

```
## Not run:
 ppm(nztrees ~ x, Strauss(13), correction="periodic")

## End(Not run)

 # Fit the nonstationary Strauss process with interaction range r=13
 # and exp(first order potential) =  activity = beta(x,y) = exp(a+bx)
 # using the periodic correction.


# Compare Maximum Pseudolikelihood, Huang-Ogata and Variational Bayes fits:
## Not run: ppm(swedishpines ~ 1, Strauss(9))

## Not run: ppm(swedishpines ~ 1, Strauss(9), method="ho")


ppm(swedishpines ~ 1, Strauss(9), method="VBlogi")

 # COVARIATES
 #
 X <- rpoispp(42)
 weirdfunction <- function(x,y){ 10 * x^2 + 5 * sin(10 * y) }
 #
 # (a) covariate values as function
 ppm(X ~ y + weirdfunction)
 #
 # (b) covariate values in pixel image
 Zimage <- as.im(weirdfunction, unit.square())
 ppm(X ~ y + Z, covariates=list(Z=Zimage))
 #
 # (c) covariate values in data frame
 Q <- quadscheme(X)
 xQ <- x.quad(Q)
 yQ <- y.quad(Q)
 Zvalues <- weirdfunction(xQ,yQ)
 ppm(Q ~  y + Z, data=data.frame(Z=Zvalues))
 # Note Q not X

 # COVARIATE FUNCTION WITH EXTRA ARGUMENTS
 #
f <- function(x,y,a){ y - a }
ppm(X ~ x + f, covfunargs=list(a=1/2))

 # COVARIATE: inside/outside window
 b <- owin(c(0.1, 0.6), c(0.1, 0.9))
 ppm(X ~ b)

 ## MULTITYPE POINT PROCESSES ###
 # fit stationary marked Poisson process
 # with different intensity for each species
## Not run: ppm(lansing ~  marks, Poisson())


 # fit nonstationary marked Poisson process
 # with different log-cubic trend for each species
## Not run: ppm(lansing ~  marks * polynom(x,y,3), Poisson())
```

ppm.object *Class of Fitted Point Process Models*

Description

A class ppm to represent a fitted stochastic model for a point process. The output of ppm.

Details

An object of class ppm represents a stochastic point process model that has been fitted to a point pattern dataset. Typically it is the output of the model fitter, ppm.

The class ppm has methods for the following standard generic functions:

generic	method	description
print	print.ppm	print details
plot	plot.ppm	plot fitted model
predict	predict.ppm	fitted intensity and conditional intensity
fitted	fitted.ppm	fitted intensity
coef	coef.ppm	fitted coefficients of model
anova	anova.ppm	Analysis of Deviance
formula	formula.ppm	Extract model formula
terms	terms.ppm	Terms in the model formula
labels	labels.ppm	Names of estimable terms in the model formula
residuals	residuals.ppm	Point process residuals
simulate	simulate.ppm	Simulate the fitted model
update	update.ppm	Change or refit the model
vcov	vcov.ppm	Variance/covariance matrix of parameter estimates
model.frame	model.frame.ppm	Model frame
model.matrix	model.matrix.ppm	Design matrix
logLik	logLik.ppm	log *pseudo* likelihood
extractAIC	extractAIC.ppm	pseudolikelihood counterpart of AIC
nobs	nobs.ppm	number of observations

Objects of class ppm can also be handled by the following standard functions, without requiring a special method:

name	description
confint	Confidence intervals for parameters
step	Stepwise model selection
drop1	One-step model improvement
add1	One-step model improvement

The class ppm also has methods for the following generic functions defined in the **spatstat** package:

generic	method	description
as.interact	as.interact.ppm	Interpoint interaction structure
as.owin	as.owin.ppm	Observation window of data

berman.test	berman.test.ppm	Berman's test
envelope	envelope.ppm	Simulation envelopes
fitin	fitin.ppm	Fitted interaction
is.marked	is.marked.ppm	Determine whether the model is marked
is.multitype	is.multitype.ppm	Determine whether the model is multitype
is.poisson	is.poisson.ppm	Determine whether the model is Poisson
is.stationary	is.stationary.ppm	Determine whether the model is stationary
cdf.test	cdf.test.ppm	Spatial distribution test
quadrat.test	quadrat.test.ppm	Quadrat counting test
reach	reach.ppm	Interaction range of model
rmhmodel	rmhmodel.ppm	Model in a form that can be simulated
rmh	rmh.ppm	Perform simulation
unitname	unitname.ppm	Name of unit of length

Information about the data (to which the model was fitted) can be extracted using data.ppm, dummy.ppm and quad.ppm.

Internal format

If you really need to get at the internals, a ppm object contains at least the following entries:

coef	the fitted regular parameters (as returned by glm)
trend	the trend formula or NULL
interaction	the point process interaction family (an object of class "interact") or NULL
Q	the quadrature scheme used
maxlogpl	the maximised value of log pseudolikelihood
correction	name of edge correction method used

See ppm for explanation of these concepts. The irregular parameters (e.g. the interaction radius of the Strauss process) are encoded in the interaction entry. However see the Warnings.

Warnings

The internal representation of ppm objects may change slightly between releases of the **spatstat** package.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, coef.ppm, fitted.ppm, print.ppm, predict.ppm, plot.ppm.

Examples

```
data(cells)
fit <- ppm(cells, ~ x, Strauss(0.1), correction="periodic")
fit
coef(fit)
## Not run:
```

```
  pred <- predict(fit)

## End(Not run)
  pred <- predict(fit, ngrid=20, type="trend")
  ## Not run:
  plot(fit)

## End(Not run)
```

---

ppm.ppp *Fit Point Process Model to Point Pattern Data*

Description

Fits a point process model to an observed point pattern.

Usage

```
   ## S3 method for class 'ppp'
ppm(Q, trend=~1, interaction=Poisson(),
       ...,
       covariates=data,
       data=NULL,
       covfunargs = list(),
       subset,
       clipwin,
       correction="border",
       rbord=reach(interaction),
       use.gam=FALSE,
       method="mpl",
       forcefit=FALSE,
       emend=project,
       project=FALSE,
       prior.mean = NULL,
       prior.var = NULL,
       nd = NULL,
       eps = NULL,
       gcontrol=list(),
       nsim=100, nrmh=1e5, start=NULL, control=list(nrep=nrmh),
       verb=TRUE,
       callstring=NULL)

   ## S3 method for class 'quad'
ppm(Q, trend=~1, interaction=Poisson(),
       ...,
       covariates=data,
       data=NULL,
       covfunargs = list(),
       subset,
       clipwin,
       correction="border",
       rbord=reach(interaction),
```

```
        use.gam=FALSE,
        method="mpl",
        forcefit=FALSE,
        emend=project,
        project=FALSE,
        prior.mean = NULL,
        prior.var = NULL,
        nd = NULL,
        eps = NULL,
        gcontrol=list(),
        nsim=100, nrmh=1e5, start=NULL, control=list(nrep=nrmh),
        verb=TRUE,
        callstring=NULL)
```

Arguments

`Q` A data point pattern (of class "ppp") to which the model will be fitted, or a quadrature scheme (of class "quad") containing this pattern.

`trend` An R formula object specifying the spatial trend to be fitted. The default formula, ~1, indicates the model is stationary and no trend is to be fitted.

`interaction` An object of class "interact" describing the point process interaction structure, or a function that makes such an object, or NULL indicating that a Poisson process (stationary or nonstationary) should be fitted.

`...` Ignored.

`data,covariates`
The values of any spatial covariates (other than the Cartesian coordinates) required by the model. Either a data frame, or a list whose entries are images, functions, windows, tessellations or single numbers. See Details.

`subset` Optional. An expression (which may involve the names of the Cartesian coordinates x and y and the names of entries in data) defining a subset of the spatial domain, to which the likelihood or pseudolikelihood should be restricted. See Details. The result of evaluating the expression should be either a logical vector, or a window (object of class "owin") or a logical-valued pixel image (object of class "im").

`clipwin` Optional. A spatial window (object of class "owin") to which data will be restricted, before model-fitting is performed. See Details.

`covfunargs` A named list containing the values of any additional arguments required by covariate functions.

`correction` The name of the edge correction to be used. The default is "border" indicating the border correction. Other possibilities may include "Ripley", "isotropic", "periodic", "translate" and "none", depending on the interaction.

`rbord` If correction = "border" this argument specifies the distance by which the window should be eroded for the border correction.

`use.gam` Logical flag; if TRUE then computations are performed using gam instead of glm.

`method` The method used to fit the model. Options are "mpl" for the method of Maximum PseudoLikelihood, "logi" for the Logistic Likelihood method, "VBlogi" for the Variational Bayes Logistic Likelihood method, and "ho" for the Huang-Ogata approximate maximum likelihood method.

forcefit	Logical flag for internal use. If forcefit=FALSE, some trivial models will be fitted by a shortcut. If forcefit=TRUE, the generic fitting method will always be used.
emend,project	(These are equivalent: project is an older name for emend.) Logical value. Setting emend=TRUE will ensure that the fitted model is always a valid point process by applying emend.ppm.
prior.mean	Optional vector of prior means for canonical parameters (for method="VBlogi"). See Details.
prior.var	Optional prior variance covariance matrix for canonical parameters (for method="VBlogi"). See Details.
nd	Optional. Integer or pair of integers. The dimension of the grid of dummy points (nd * nd or nd[1] * nd[2]) used to evaluate the integral in the pseudolikelihood. Incompatible with eps.
eps	Optional. A positive number, or a vector of two positive numbers, giving the horizontal and vertical spacing, respectively, of the grid of dummy points. Incompatible with nd.
gcontrol	Optional. List of parameters passed to glm.control (or passed to gam.control if use.gam=TRUE) controlling the model-fitting algorithm.
nsim	Number of simulated realisations to generate (for method="ho")
nrmh	Number of Metropolis-Hastings iterations for each simulated realisation (for method="ho")
start,control	Arguments passed to rmh controlling the behaviour of the Metropolis-Hastings algorithm (for method="ho")
verb	Logical flag indicating whether to print progress reports (for method="ho")
callstring	Internal use only.

Details

NOTE: This help page describes the **old syntax** of the function ppm, described in many older documents. This old syntax is still supported. However, if you are learning about ppm for the first time, we recommend you use the **new syntax** described in the help file for ppm.

This function fits a point process model to an observed point pattern. The model may include spatial trend, interpoint interaction, and dependence on covariates.

basic use: In basic use, Q is a point pattern dataset (an object of class "ppp") to which we wish to fit a model.

The syntax of ppm() is closely analogous to the R functions glm and gam. The analogy is:

glm	**ppm**
formula	trend
family	interaction

The point process model to be fitted is specified by the arguments trend and interaction which are respectively analogous to the formula and family arguments of glm().

Systematic effects (spatial trend and/or dependence on spatial covariates) are specified by the argument trend. This is an R formula object, which may be expressed in terms of the Cartesian coordinates x, y, the marks marks, or the variables in covariates (if supplied), or both. It specifies the **logarithm** of the first order potential of the process. The formula should not use any names beginning with .mpl as these are reserved for internal use. If trend is absent or equal to the default, `~1`, then the model to be fitted is stationary (or at least, its first order potential is constant).

The symbol `.` in the trend expression stands for all the covariates supplied in the argument `data`. For example the formula `~ .` indicates an additive model with a main effect for each covariate in `data`.

Stochastic interactions between random points of the point process are defined by the argument `interaction`. This is an object of class `"interact"` which is initialised in a very similar way to the usage of family objects in `glm` and `gam`. The models currently available are: `AreaInter`, `BadGey`, `Concom`, `DiggleGatesStibbard`, `DiggleGratton`, `Fiksel`, `Geyer`, `Hardcore`, `Hybrid`, `LennardJones`, `MultiStrauss`, `MultiStraussHard`, `OrdThresh`, `Ord`, `Pairwise`, `PairPiece`, `Penttinen`, `Poisson`, `Saturated`, `SatPiece`, `Softcore`, `Strauss`, `StraussHard` and `Triplets`. See the examples below. It is also possible to combine several interactions using `Hybrid`.

If `interaction` is missing or `NULL`, then the model to be fitted has no interpoint interactions, that is, it is a Poisson process (stationary or nonstationary according to `trend`). In this case the methods of maximum pseudolikelihood and maximum logistic likelihood coincide with maximum likelihood.

The fitted point process model returned by this function can be printed (by the print method `print.ppm`) to inspect the fitted parameter values. If a nonparametric spatial trend was fitted, this can be extracted using the predict method `predict.ppm`.

Models with covariates: To fit a model involving spatial covariates other than the Cartesian coordinates x and y, the values of the covariates should be supplied in the argument `covariates`. Note that it is not sufficient to have observed the covariate only at the points of the data point pattern; the covariate must also have been observed at other locations in the window.

Typically the argument `covariates` is a list, with names corresponding to variables in the `trend` formula. Each entry in the list is either

a pixel image, giving the values of a spatial covariate at a fine grid of locations. It should be an object of class `"im"`, see `im.object`.

a function, which can be evaluated at any location `(x,y)` to obtain the value of the spatial covariate. It should be a `function(x, y)` or `function(x, y, ...)` in the R language. The first two arguments of the function should be the Cartesian coordinates x and y. The function may have additional arguments; if the function does not have default values for these additional arguments, then the user must supply values for them, in `covfunargs`. See the Examples.

a window, interpreted as a logical variable which is `TRUE` inside the window and `FALSE` outside it. This should be an object of class `"owin"`.

a tessellation, interpreted as a factor covariate. For each spatial location, the factor value indicates which tile of the tessellation it belongs to. This should be an object of class `"tess"`.

a single number, indicating a covariate that is constant in this dataset.

The software will look up the values of each covariate at the required locations (quadrature points).

Note that, for covariate functions, only the *name* of the function appears in the trend formula. A covariate function is treated as if it were a single variable. The function arguments do not appear in the trend formula. See the Examples.

If `covariates` is a list, the list entries should have names corresponding to the names of covariates in the model formula `trend`. The variable names `x`, `y` and `marks` are reserved for the Cartesian coordinates and the mark values, and these should not be used for variables in `covariates`.

If covariates is a data frame, Q must be a quadrature scheme (see under Quadrature Schemes below). Then covariates must have as many rows as there are points in Q. The ith row of covariates should contain the values of spatial variables which have been observed at the ith point of Q.

Quadrature schemes: In advanced use, Q may be a 'quadrature scheme'. This was originally just a technicality but it has turned out to have practical uses, as we explain below.

Quadrature schemes are required for our implementation of the method of maximum pseudolikelihood. The definition of the pseudolikelihood involves an integral over the spatial window containing the data. In practice this integral must be approximated by a finite sum over a set of quadrature points. We use the technique of Baddeley and Turner (2000), a generalisation of the Berman-Turner (1992) device. In this technique the quadrature points for the numerical approximation include all the data points (points of the observed point pattern) as well as additional 'dummy' points.

Quadrature schemes are also required for the method of maximum logistic likelihood, which combines the data points with additional 'dummy' points.

A quadrature scheme is an object of class "quad" (see quad.object) which specifies both the data point pattern and the dummy points for the quadrature scheme, as well as the quadrature weights associated with these points. If Q is simply a point pattern (of class "ppp", see ppp.object) then it is interpreted as specifying the data points only; a set of dummy points specified by default.dummy() is added, and the default weighting rule is invoked to compute the quadrature weights.

Finer quadrature schemes (i.e. those with more dummy points) generally yield a better approximation, at the expense of higher computational load.

An easy way to fit models using a finer quadrature scheme is to let Q be the original point pattern data, and use the argument nd to determine the number of dummy points in the quadrature scheme.

Complete control over the quadrature scheme is possible. See quadscheme for an overview. Use quadscheme(X, D, method="dirichlet") to compute quadrature weights based on the Dirichlet tessellation, or quadscheme(X, D, method="grid") to compute quadrature weights by counting points in grid squares, where X and D are the patterns of data points and dummy points respectively. Alternatively use pixelquad to make a quadrature scheme with a dummy point at every pixel in a pixel image.

A practical advantage of quadrature schemes arises when we want to fit a model involving covariates (e.g. soil pH). Suppose we have only been able to observe the covariates at a small number of locations. Suppose cov.dat is a data frame containing the values of the covariates at the data points (i.e.\ cov.dat[i,] contains the observations for the ith data point) and cov.dum is another data frame (with the same columns as cov.dat) containing the covariate values at another set of points whose locations are given by the point pattern Y. Then setting Q = quadscheme(X,Y) combines the data points and dummy points into a quadrature scheme, and covariates = rbind(cov.dat, cov.dum) combines the covariate data frames. We can then fit the model by calling ppm(Q, ..., covariates).

Model-fitting technique: There are several choices for the technique used to fit the model.

method="mpl" (the default): the model will be fitted by maximising the pseudolikelihood (Besag, 1975) using the Berman-Turner computational approximation (Berman and Turner, 1992; Baddeley and Turner, 2000). Maximum pseudolikelihood is equivalent to maximum likelihood if the model is a Poisson process. Maximum pseudolikelihood is biased if the interpoint interaction is very strong, unless there is a large number of dummy points. The default settings for method='mpl' specify a moderately large number of dummy points, striking a compromise between speed and accuracy.

method="logi": the model will be fitted by maximising the logistic likelihood (Baddeley et al, 2014). This technique is roughly equivalent in speed to maximum pseudolikelihood, but is believed to be less biased. Because it is less biased, the default settings for `method='logi'` specify a relatively small number of dummy points, so that this method is the fastest, in practice.

method="VBlogi": the model will be fitted in a Bayesian setup by maximising the posterior probability density for the canonical model parameters. This uses the variational Bayes approximation to the posterior derived from the logistic likelihood as described in Rajala (2014). The prior is assumed to be multivariate Gaussian with mean vector `prior.mean` and variance-covariance matrix `prior.var`.

method="ho": the model will be fitted by applying the approximate maximum likelihood method of Huang and Ogata (1999). See below. The Huang-Ogata method is slower than the other options, but has better statistical properties.

Note that `method='logi'`, `method='VBlogi'` and `method='ho'` involve randomisation, so that the results are subject to random variation.

Huang-Ogata method: If `method="ho"` then the model will be fitted using the Huang-Ogata (1999) approximate maximum likelihood method. First the model is fitted by maximum pseudolikelihood as described above, yielding an initial estimate of the parameter vector θ_0. From this initial model, `nsim` simulated realisations are generated. The score and Fisher information of the model at $\theta = \theta_0$ are estimated from the simulated realisations. Then one step of the Fisher scoring algorithm is taken, yielding an updated estimate θ_1. The corresponding model is returned.

Simulated realisations are generated using `rmh`. The iterative behaviour of the Metropolis-Hastings algorithm is controlled by the arguments `start` and `control` which are passed to `rmh`.

As a shortcut, the argument `nrmh` determines the number of Metropolis-Hastings iterations run to produce one simulated realisation (if `control` is absent). Also if `start` is absent or equal to `NULL`, it defaults to `list(n.start=N)` where `N` is the number of points in the data point pattern.

Edge correction Edge correction should be applied to the sufficient statistics of the model, to reduce bias. The argument `correction` is the name of an edge correction method. The default `correction="border"` specifies the border correction, in which the quadrature window (the domain of integration of the pseudolikelihood) is obtained by trimming off a margin of width `rbord` from the observation window of the data pattern. Not all edge corrections are implemented (or implementable) for arbitrary windows. Other options depend on the argument `interaction`, but these generally include `correction="periodic"` (the periodic or toroidal edge correction in which opposite edges of a rectangular window are identified) and `correction="translate"` (the translation correction, see Baddeley 1998 and Baddeley and Turner 2000). For pairwise interaction models there is also Ripley's isotropic correction, identified by `correction="isotropic"` or `"Ripley"`.

Subsetting The arguments `subset` and `clipwin` specify that the model should be fitted to a restricted subset of the available data. These arguments are equivalent for Poisson point process models, but different for Gibbs models. If `clipwin` is specified, then all the available data will be restricted to this spatial region, and data outside this region will be discarded, before the model is fitted. If `subset` is specified, then no data are deleted, but the domain of integration of the likelihood or pseudolikelihood is restricted to the `subset`. For Poisson models, these two arguments have the same effect; but for a Gibbs model, interactions between points inside and outside the `subset` are taken into account, while interactions between points inside and outside the `clipwin` are ignored.

Value

An object of class `"ppm"` describing a fitted point process model.

See `ppm.object` for details of the format of this object and methods available for manipulating it.

Interaction parameters

Apart from the Poisson model, every point process model fitted by `ppm` has parameters that determine the strength and range of 'interaction' or dependence between points. These parameters are of two types:

regular parameters: A parameter ϕ is called *regular* if the log likelihood is a linear function of θ where $\theta = \theta(\psi)$ is some transformation of ψ. [Then θ is called the canonical parameter.]

irregular parameters Other parameters are called *irregular*.

Typically, regular parameters determine the 'strength' of the interaction, while irregular parameters determine the 'range' of the interaction. For example, the Strauss process has a regular parameter γ controlling the strength of interpoint inhibition, and an irregular parameter r determining the range of interaction.

The `ppm` command is only designed to estimate regular parameters of the interaction. It requires the values of any irregular parameters of the interaction to be fixed. For example, to fit a Strauss process model to the `cells` dataset, you could type `ppm(cells, ~1, Strauss(r=0.07))`. Note that the value of the irregular parameter `r` must be given. The result of this command will be a fitted model in which the regular parameter γ has been estimated.

To determine the irregular parameters, there are several practical techniques, but no general statistical theory available. Useful techniques include maximum profile pseudolikelihood, which is implemented in the command `profilepl`, and Newton-Raphson maximisation, implemented in the experimental command `ippm`.

Some irregular parameters can be estimated directly from data: the hard-core radius in the model `Hardcore` and the matrix of hard-core radii in `MultiHard` can be estimated easily from data. In these cases, `ppm` allows the user to specify the interaction without giving the value of the irregular parameter. The user can give the hard core interaction as `interaction=Hardcore()` or even `interaction=Hardcore`, and the hard core radius will then be estimated from the data.

Error and Warning Messages

Some common error messages and warning messages are listed below, with explanations.

"System is computationally singular" The Fisher information matrix of the fitted model has a determinant close to zero, so that the matrix cannot be inverted, and the software cannot calculate standard errors or confidence intervals. This error is usually reported when the model is printed, because the `print` method calculates standard errors for the fitted parameters. Singularity usually occurs because the spatial coordinates in the original data were very large numbers (e.g. expressed in metres) so that the fitted coefficients were very small numbers. The simple remedy is to **rescale the data**, for example, to convert from metres to kilometres by `X <- rescale(X, 1000)`, then re-fit the model. Singularity can also occur if the covariate values are very large numbers, or if the covariates are approximately collinear.

"Covariate values were NA or undefined at X% (M out of N) of the quadrature points" The covariate data (typically a pixel image) did not provide values of the covariate at some of the spatial locations in the observation window of the point pattern. This means that the spatial domain of the pixel image does not completely cover the observation window of the point pattern. If the percentage is small, this warning can be ignored - typically it happens because of rounding effects which cause the pixel image to be one-pixel-width narrower than the observation window. However if more than a few percent of covariate values are undefined, it would be prudent to check that the pixel images are correct, and are correctly registered in their spatial relation to the observation window.

"Model is unidentifiable" It is not possible to estimate all the model parameters from this dataset. The error message gives a further explanation, such as "data pattern is empty". Choose a simpler model, or check the data.

"N data points are illegal (zero conditional intensity)" In a Gibbs model (i.e. with interaction between points), the conditional intensity may be zero at some spatial locations, indicating that the model forbids the presence of a point at these locations. However if the conditional intensity is zero *at a data point*, this means that the model is inconsistent with the data. Modify the interaction parameters so that the data point is not illegal (e.g. reduce the value of the hard core radius) or choose a different interaction.

Warnings

The implementation of the Huang-Ogata method is experimental; several bugs were fixed in **spatstat** 1.19-0.

See the comments above about the possible inefficiency and bias of the maximum pseudolikelihood estimator.

The accuracy of the Berman-Turner approximation to the pseudolikelihood depends on the number of dummy points used in the quadrature scheme. The number of dummy points should at least equal the number of data points.

The parameter values of the fitted model do not necessarily determine a valid point process. Some of the point process models are only defined when the parameter values lie in a certain subset. For example the Strauss process only exists when the interaction parameter γ is less than or equal to 1, corresponding to a value of `ppm()$theta[2]` less than or equal to `0`.

By default (if `emend=FALSE`) the algorithm maximises the pseudolikelihood without constraining the parameters, and does not apply any checks for sanity after fitting the model. This is because the fitted parameter value could be useful information for data analysis. To constrain the parameters to ensure that the model is a valid point process, set `emend=TRUE`. See also the functions `valid.ppm` and `emend.ppm`.

The `trend` formula should not use any variable names beginning with the prefixes `.mpl` or `Interaction` as these names are reserved for internal use. The data frame `covariates` should have as many rows as there are points in `Q`. It should not contain variables called `x`, `y` or `marks` as these names are reserved for the Cartesian coordinates and the marks.

If the model formula involves one of the functions `poly()`, `bs()` or `ns()` (e.g. applied to spatial coordinates `x` and `y`), the fitted coefficients can be misleading. The resulting fit is not to the raw spatial variates (`x`, `x^2`, `x*y`, etc.) but to a transformation of these variates. The transformation is implemented by `poly()` in order to achieve better numerical stability. However the resulting coefficients are appropriate for use with the transformed variates, not with the raw variates. This affects the interpretation of the constant term in the fitted model, `logbeta`. Conventionally, β is the background intensity, i.e. the value taken by the conditional intensity function when all predictors (including spatial or "trend" predictors) are set equal to 0. However the coefficient actually produced is the value that the log conditional intensity takes when all the predictors, including the *transformed* spatial predictors, are set equal to `0`, which is not the same thing.

Worse still, the result of `predict.ppm` can be completely wrong if the trend formula contains one of the functions `poly()`, `bs()` or `ns()`. This is a weakness of the underlying function `predict.glm`.

If you wish to fit a polynomial trend, we offer an alternative to `poly()`, namely `polynom()`, which avoids the difficulty induced by transformations. It is completely analogous to `poly` except that it does not orthonormalise. The resulting coefficient estimates then have their natural interpretation and can be predicted correctly. Numerical stability may be compromised.

Values of the maximised pseudolikelihood are not comparable if they have been obtained with different values of `rbord`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A., Coeurjolly, J.-F., Rubak, E. and Waagepetersen, R. (2014) Logistic regression for spatial Gibbs point processes. *Biometrika* **101** (2) 377–392.

Baddeley, A. and Turner, R. Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** (2000) 283–322.

Berman, M. and Turner, T.R. Approximating point process likelihoods with GLIM. *Applied Statistics* **41** (1992) 31–38.

Besag, J. Statistical analysis of non-lattice data. *The Statistician* **24** (1975) 179-195.

Diggle, P.J., Fiksel, T., Grabarnik, P., Ogata, Y., Stoyan, D. and Tanemura, M. On parameter estimation for pairwise interaction processes. *International Statistical Review* **62** (1994) 99-117.

Huang, F. and Ogata, Y. Improvements of the maximum pseudo-likelihood estimators in various spatial statistical models. *Journal of Computational and Graphical Statistics* **8** (1999) 510-530.

Jensen, J.L. and Moeller, M. Pseudolikelihood for exponential family models of spatial point processes. *Annals of Applied Probability* **1** (1991) 445–461.

Jensen, J.L. and Kuensch, H.R. On asymptotic normality of pseudo likelihood estimates for pairwise interaction processes, *Annals of the Institute of Statistical Mathematics* **46** (1994) 475-486.

Rajala T. (2014) *A note on Bayesian logistic regression for spatial exponential family Gibbs point processes*, Preprint on ArXiv.org. http://arxiv.org/abs/1411.0539

See Also

`ppm.object` for details of how to print, plot and manipulate a fitted model.

`ppp` and `quadscheme` for constructing data.

Interactions: `AreaInter`, `BadGey`, `Concom`, `DiggleGatesStibbard`, `DiggleGratton`, `Fiksel`, `Geyer`, `Hardcore`, `Hybrid`, `LennardJones`, `MultiStrauss`, `MultiStraussHard`, `OrdThresh`, `Ord`, `Pairwise`, `PairPiece`, `Penttinen`, `Poisson`, `Saturated`, `SatPiece`, `Softcore`, `Strauss`, `StraussHard` and `Triplets`.

See `profilepl` for advice on fitting nuisance parameters in the interaction, and `ippm` for irregular parameters in the trend.

See `valid.ppm` and `emend.ppm` for ensuring the fitted model is a valid point process.

Examples

```
 # fit the stationary Poisson process
 # to point pattern 'nztrees'

 ppm(nztrees)
 ppm(nztrees ~ 1)

 ## Not run:
 Q <- quadscheme(nztrees)
 ppm(Q)
 # equivalent.

## End(Not run)
```

```
 ## Not run:
  ppm(nztrees, nd=128)

## End(Not run)


fit1 <- ppm(nztrees, ~ x)
 # fit the nonstationary Poisson process
 # with intensity function lambda(x,y) = exp(a + bx)
 # where x,y are the Cartesian coordinates
 # and a,b are parameters to be estimated

fit1
coef(fit1)
coef(summary(fit1))

## Not run:
 ppm(nztrees, ~ polynom(x,2))

## End(Not run)


 # fit the nonstationary Poisson process
 # with intensity function lambda(x,y) = exp(a + bx + cx^2)

 ## Not run:
 library(splines)
 ppm(nztrees, ~ bs(x,df=3))

## End(Not run)
 #       WARNING: do not use predict.ppm() on this result
 # Fits the nonstationary Poisson process
 # with intensity function lambda(x,y) = exp(B(x))
 # where B is a B-spline with df = 3

## Not run:
 ppm(nztrees, ~1, Strauss(r=10), rbord=10)

## End(Not run)

 # Fit the stationary Strauss process with interaction range r=10
 # using the border method with margin rbord=10

## Not run:
 ppm(nztrees, ~ x, Strauss(13), correction="periodic")

## End(Not run)

 # Fit the nonstationary Strauss process with interaction range r=13
 # and exp(first order potential) =  activity = beta(x,y) = exp(a+bx)
 # using the periodic correction.


# Compare Maximum Pseudolikelihood, Huang-Ogata and VB fits:
## Not run: ppm(swedishpines, ~1, Strauss(9))
```

```
## Not run: ppm(swedishpines, ~1, Strauss(9), method="ho")


ppm(swedishpines, ~1, Strauss(9), method="VBlogi")

 # COVARIATES
 #
 X <- rpoispp(42)
 weirdfunction <- function(x,y){ 10 * x^2 + 5 * sin(10 * y) }
 #
 # (a) covariate values as function
 ppm(X, ~ y + Z, covariates=list(Z=weirdfunction))
 #
 # (b) covariate values in pixel image
 Zimage <- as.im(weirdfunction, unit.square())
 ppm(X, ~ y + Z, covariates=list(Z=Zimage))
 #
 # (c) covariate values in data frame
 Q <- quadscheme(X)
 xQ <- x.quad(Q)
 yQ <- y.quad(Q)
 Zvalues <- weirdfunction(xQ,yQ)
 ppm(Q, ~ y + Z, covariates=data.frame(Z=Zvalues))
 # Note Q not X

 # COVARIATE FUNCTION WITH EXTRA ARGUMENTS
 #
f <- function(x,y,a){ y - a }
ppm(X, ~x + f, covariates=list(f=f), covfunargs=list(a=1/2))

 # COVARIATE: inside/outside window
 b <- owin(c(0.1, 0.6), c(0.1, 0.9))
 ppm(X, ~w, covariates=list(w=b))

 ## MULTITYPE POINT PROCESSES ###
 # fit stationary marked Poisson process
 # with different intensity for each species
## Not run: ppm(lansing, ~ marks, Poisson())


 # fit nonstationary marked Poisson process
 # with different log-cubic trend for each species
## Not run: ppm(lansing, ~ marks * polynom(x,y,3), Poisson())
```

---

ppmInfluence *Leverage and Influence Measures for Spatial Point Process Model*

Description

Calculates all the leverage and influence measures described in influence.ppm, leverage.ppm and dfbetas.ppm.

Usage

```
ppmInfluence(fit,
             what = c("leverage", "influence", "dfbetas"),
             ...,
             iScore = NULL, iHessian = NULL, iArgs = NULL,
             drop = FALSE,
             fitname = NULL)
```

Arguments

`fit`	A fitted point process model of class "ppm".
`what`	Character vector specifying which quantities are to be calculated. Default is to calculate all quantities.
`...`	Ignored.
`iScore,iHessian`	Components of the score vector and Hessian matrix for the irregular parameters, if required. See Details.
`iArgs`	List of extra arguments for the functions `iScore`, `iHessian` if required.
`drop`	Logical. Whether to include (`drop=FALSE`) or exclude (`drop=TRUE`) contributions from quadrature points that were not used to fit the model.
`fitname`	Optional character string name for the fitted model `fit`.

Details

This function calculates all the leverage and influence measures described in `influence.ppm`, `leverage.ppm` and `dfbetas.ppm`.

When analysing large datasets, the user can call `ppmInfluence` to perform the calculations efficiently, then extract the leverage and influence values as desired. For example the leverage can be extracted either as `result$leverage` or `leverage(result)`.

If the point process model trend has irregular parameters that were fitted (using `ippm`) then the influence calculation requires the first and second derivatives of the log trend with respect to the irregular parameters. The argument `iScore` should be a list, with one entry for each irregular parameter, of R functions that compute the partial derivatives of the log trend (i.e. log intensity or log conditional intensity) with respect to each irregular parameter. The argument `iHessian` should be a list, with p^2 entries where p is the number of irregular parameters, of R functions that compute the second order partial derivatives of the log trend with respect to each pair of irregular parameters.

Value

A list containing the leverage and influence measures specified by `what`. The result also belongs to the class "ppmInfluence".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

`leverage.ppm`, `influence.ppm`, `dfbetas.ppm`

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X ~ x+y)
fI <- ppmInfluence(fit)
fI$influence
influence(fI)
fI$leverage
fI$dfbetas
```

---

ppp *Create a Point Pattern*

Description

Creates an object of class "ppp" representing a point pattern dataset in the two-dimensional plane.

Usage

```
ppp(x,y, ..., window, marks,
    check=TRUE, checkdup=check, drop=TRUE)
```

Arguments

x	Vector of x coordinates of data points
y	Vector of y coordinates of data points
window	window of observation, an object of class "owin"
...	arguments passed to owin to create the window, if window is missing
marks	(optional) mark values for the points. A vector or data frame.
check	Logical value indicating whether to check that all the (x, y) points lie inside the specified window. Do not set this to FALSE unless you are absolutely sure that this check is unnecessary. See Warnings below.
checkdup	Logical value indicating whether to check for duplicated coordinates. See Warnings below.
drop	Logical flag indicating whether to simplify data frames of marks. See Details.

Details

In the **spatstat** library, a point pattern dataset is described by an object of class "ppp". This function creates such objects.

The vectors x and y must be numeric vectors of equal length. They are interpreted as the cartesian coordinates of the points in the pattern. Note that x and y are permitted to have length zero, corresponding to an empty point pattern; this is the default if these arguments are missing.

A point pattern dataset is assumed to have been observed within a specific region of the plane called the observation window. An object of class "ppp" representing a point pattern contains information specifying the observation window. This window must always be specified when creating a point pattern dataset; there is intentionally no default action of “guessing” the window dimensions from the data points alone.

You can specify the observation window in several (mutually exclusive) ways:

- xrange, yrange specify a rectangle with these dimensions;
- poly specifies a polygonal boundary. If the boundary is a single polygon then poly must be a list with components x,y giving the coordinates of the vertices. If the boundary consists of several disjoint polygons then poly must be a list of such lists so that poly[[i]]$x gives the x coordinates of the vertices of the ith boundary polygon.
- mask specifies a binary pixel image with entries that are TRUE if the corresponding pixel is inside the window.
- window is an object of class "owin" specifying the window. A window object can be created by owin from raw coordinate data. Special shapes of windows can be created by the functions square, hexagon, regularpolygon, disc and ellipse. See the Examples.

The arguments xrange, yrange or poly or mask are passed to the window creator function owin for interpretation. See owin for further details.

The argument window, if given, must be an object of class "owin". It is a full description of the window geometry, and could have been obtained from owin or as.owin, or by just extracting the observation window of another point pattern, or by manipulating such windows. See owin or the Examples below.

The points with coordinates x and y **must** lie inside the specified window, in order to define a valid object of this class. Any points which do not lie inside the window will be removed from the point pattern, and a warning will be issued. See the section on Rejected Points.

The name of the unit of length for the x and y coordinates can be specified in the dataset, using the argument unitname, which is passed to owin. See the examples below, or the help file for owin.

The optional argument marks is given if the point pattern is marked, i.e. if each data point carries additional information. For example, points which are classified into two or more different types, or colours, may be regarded as having a mark which identifies which colour they are. Data recording the locations and heights of trees in a forest can be regarded as a marked point pattern where the mark is the tree height.

The argument marks can be either

- a vector, of the same length as x and y, which is interpreted so that marks[i] is the mark attached to the point (x[i],y[i]). If the mark is a real number then marks should be a numeric vector, while if the mark takes only a finite number of possible values (e.g. colours or types) then marks should be a factor.
- a data frame, with the number of rows equal to the number of points in the point pattern. The ith row of the data frame is interpreted as containing the mark values for the ith point in the point pattern. The columns of the data frame correspond to different mark variables (e.g. tree species and tree diameter).

If drop=TRUE (the default), then a data frame with only one column will be converted to a vector, and a data frame with no columns will be converted to NULL.

See ppp.object for a description of the class "ppp".

Users would normally invoke ppp to create a point pattern, but the functions as.ppp and scanpp may sometimes be convenient.

Value

An object of class "ppp" describing a point pattern in the two-dimensional plane (see ppp.object).

Invalid coordinate values

The coordinate vectors x and y must contain only finite numerical values. If the coordinates include any of the values NA, NaN, Inf or -Inf, these will be removed.

Rejected points

The points with coordinates x and y **must** lie inside the specified window, in order to define a valid object of class "ppp". Any points which do not lie inside the window will be removed from the point pattern, and a warning will be issued.

The rejected points are still accessible: they are stored as an attribute of the point pattern called "rejects" (which is an object of class "ppp" containing the rejected points in a large window). However, rejected points in a point pattern will be ignored by all other functions except plot.ppp.

To remove the rejected points altogether, use as.ppp. To include the rejected points, you will need to find a larger window that contains them, and use this larger window in a call to ppp.

Warnings

The code will check for problems with the data, and issue a warning if any problems are found. The checks and warnings can be switched off, for efficiency's sake, but this should only be done if you are confident that the data do not have these problems.

Setting check=FALSE will disable all the checking procedures: the check for points outside the window, and the check for duplicated points. This is extremely dangerous, because points lying outside the window will break many of the procedures in **spatstat**, causing crashes and strange errors. Set check=FALSE only if you are absolutely sure that there are no points outside the window.

If duplicated points are found, a warning is issued, but no action is taken. Duplicated points are not illegal, but may cause unexpected problems later. Setting checkdup=FALSE will disable the check for duplicated points. Do this only if you already know the answer.

Methodology and software for spatial point patterns often assume that all points are distinct so that there are no duplicated points. If duplicated points are present, the consequence could be an incorrect result or a software crash. To the best of our knowledge, all **spatstat** code handles duplicated points correctly. However, if duplicated points are present, we advise using unique.ppp or multiplicity.ppp to eliminate duplicated points and re-analyse the data.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppp.object, as.ppp, owin.object, owin, as.owin

Examples

```
# some arbitrary coordinates in [0,1]
x <- runif(20)
y <- runif(20)

# the following are equivalent
X <- ppp(x, y, c(0,1), c(0,1))
X <- ppp(x, y)
X <- ppp(x, y, window=owin(c(0,1),c(0,1)))

# specify that the coordinates are given in metres
X <- ppp(x, y, c(0,1), c(0,1), unitname=c("metre","metres"))

## Not run: plot(X)
```

```
# marks
m <- sample(1:2, 20, replace=TRUE)
m <- factor(m, levels=1:2)
X <- ppp(x, y, c(0,1), c(0,1), marks=m)
## Not run: plot(X)

# polygonal window
X <- ppp(x, y, poly=list(x=c(0,10,0), y=c(0,0,10)))
## Not run: plot(X)

# circular window of radius 2
X <- ppp(x, y, window=disc(2))

# copy the window from another pattern
data(cells)
X <- ppp(x, y, window=Window(cells))
```

---

ppp.object *Class of Point Patterns*

Description

A class "ppp" to represent a two-dimensional point pattern. Includes information about the window in which the pattern was observed. Optionally includes marks.

Details

This class represents a two-dimensional point pattern dataset. It specifies

- the locations of the points
- the window in which the pattern was observed
- optionally, "marks" attached to each point (extra information such as a type label).

If X is an object of type ppp, it contains the following elements:

x	vector of x coordinates of data points
y	vector of y coordinates of data points
n	number of points
window	window of observation (an object of class owin)
marks	optional vector or data frame of marks

Users are strongly advised not to manipulate these entries directly.

Objects of class "ppp" may be created by the function ppp and converted from other types of data by the function as.ppp. Note that you must always specify the window of observation; there is intentionally no default action of "guessing" the window dimensions from the data points alone.

Standard point pattern datasets provided with the package include amacrine, betacells, bramblecanes, cells, demopat, ganglia, lansing, longleaf, nztrees, redwood, simdat and swedishpines.

Point patterns may be scanned from your own data files by scanpp or by using read.table and as.ppp.

They may be manipulated by the functions [.ppp and superimpose.

Point pattern objects can be plotted just by typing plot(X) which invokes the plot method for point pattern objects, plot.ppp. See plot.ppp for further information.

There are also methods for summary and print for point patterns. Use summary(X) to see a useful description of the data.

Patterns may be generated at random by runifpoint, rpoispp, rMaternI, rMaternII, rSSI, rNeymanScott, rMatClust, and rThomas.

Most functions which are intended to operate on a window (of class owin) will, if presented with a ppp object instead, automatically extract the window information from the point pattern.

Warnings

The internal representation of marks is likely to change in the next release of this package.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin, ppp, as.ppp, [.ppp

Examples

```
x <- runif(100)
y <- runif(100)
X <- ppp(x, y, c(0,1),c(0,1))
X
## Not run: plot(X)
mar <- sample(1:3, 100, replace=TRUE)
mm <- ppp(x, y, c(0,1), c(0,1), marks=mar)
## Not run: plot(mm)
# points with mark equal to 2
ss <- mm[ mm$marks == 2 , ]
## Not run: plot(ss)
# left half of pattern 'mm'
lu <- owin(c(0,0.5),c(0,1))
mmleft <- mm[ , lu]
## Not run: plot(mmleft)
## Not run:
if(FALSE) {
# input data from file
qq <- scanpp("my.table", unit.square())

# interactively build a point pattern
plot(unit.square())
X <- as.ppp(locator(10), unit.square())
plot(X)
}

## End(Not run)
```

pppdist *Distance Between Two Point Patterns*

Description

Given two point patterns, find the distance between them based on optimal point matching.

Usage

```
pppdist(X, Y, type = "spa", cutoff = 1, q = 1, matching = TRUE,
  ccode = TRUE, auction = TRUE, precision = NULL, approximation = 10,
  show.rprimal = FALSE, timelag = 0)
```

Arguments

X,Y	Two point patterns (objects of class "ppp").
type	A character string giving the type of distance to be computed. One of "spa" (default), "ace" or "mat", indicating whether the algorithm should find the optimal matching based on "subpattern assignment", "assignment only if cardinalities are equal" or "mass transfer". See Details.
cutoff	The value > 0 at which interpoint distances are cut off.
q	The order of the average that is applied to the interpoint distances. May be Inf, in which case the maximum of the interpoint distances is taken.
matching	Logical. Whether to return the optimal matching or only the associated distance.
ccode	Logical. If FALSE, R code is used which allows for higher precision, but is much slower.
auction	Logical. By default a version of Bertsekas' auction algorithm is used to compute an optimal point matching if type is either "spa" or "ace". If auction is FALSE (or type is "mat") a specialized primal-dual algorithm is used instead. This was the standard in earlier versions of **spatstat**, but is several orders of magnitudes slower.
precision	Index controlling accuracy of algorithm. The q-th powers of interpoint distances will be rounded to the nearest multiple of 10^(-precision). There is a sensible default which depends on ccode.
approximation	If q = Inf, compute distance based on the optimal matching for the corresponding distance of order approximation. Can be Inf, but this makes computations extremely slow.
show.rprimal	Logical. Whether to plot the progress of the primal-dual algorithm. If TRUE, slow primal-dual R code is used, regardless of the arguments ccode and auction.
timelag	Time lag, in seconds, between successive displays of the iterative solution of the restricted primal problem.

Details

Computes the distance between point patterns X and Y based on finding the matching between them which minimizes the average of the distances between matched points (if q=1), the maximum distance between matched points (if q=Inf), and in general the q-th order average (i.e. the 1/qth power of the sum of the qth powers) of the distances between matched points. Distances between matched points are Euclidean distances cut off at the value of cutoff.

The parameter type controls the behaviour of the algorithm if the cardinalities of the point patterns are different. For the type "spa" (subpattern assignment) the subpattern of the point pattern with the larger cardinality n that is closest to the point pattern with the smaller cardinality m is determined; then the q-th order average is taken over n values: the m distances of matched points and $n - m$ "penalty distances" of value cutoff for the unmatched points. For the type "ace" (assignment only if cardinalities equal) the matching is empty and the distance returned is equal to cutoff if the cardinalities differ. For the type "mat" (mass transfer) each point pattern is assumed to have total mass m (= the smaller cardinality) distributed evenly among its points; the algorithm finds then the "mass transfer plan" that minimizes the q-th order weighted average of the distances, where the weights are given by the transferred mass divided by m. The result is a fractional matching (each match of two points has a weight in $(0, 1]$) with the minimized quantity as the associated distance.

The central problem to be solved is the assignment problem (for types "spa" and "ace") or the more general transport problem (for type "mat"). Both are well-known problems in discrete optimization, see e.g. Luenberger (2003).

For the assignment problem pppdist uses by default the forward/backward version of Bertsekas' auction algorithm with automated epsilon scaling; see Bertsekas (1992). The implemented version gives good overall performance and can handle point patterns with several thousand points.

For the transport problem a specialized primal-dual algorithm is employed; see Luenberger (2003), Section 5.9. The C implementation used by default can handle patterns with a few hundreds of points, but should not be used with thousands of points. By setting show.rprimal = TRUE, some insight in the working of the algorithm can be gained.

For a broader selection of optimal transport algorithms that are not restricted to spatial point patterns and allow for additional fine tuning, we recommend the R package **transport**.

For moderate and large values of q there can be numerical issues based on the fact that the q-th powers of distances are taken and some positive values enter the optimization algorithm as zeroes because they are too small in comparison with the larger values. In this case the number of zeroes introduced is given in a warning message, and it is possible then that the matching obtained is not optimal and the associated distance is only a strict upper bound of the true distance. As a general guideline (which can be very wrong in special situations) a small number of zeroes (up to about 50% of the smaller point pattern cardinality m) usually still results in the right matching, and the number can even be quite a bit higher and usually still provides a highly accurate upper bound for the distance. These numerical problems can be reduced by enforcing (much slower) R code via the argument ccode = FALSE.

For q = Inf there is no fast algorithm available, which is why approximation is normally used: for finding the optimal matching, q is set to the value of approximation. The resulting distance is still given as the maximum rather than the q-th order average in the corresponding distance computation. If approximation = Inf, approximation is suppressed and a very inefficient exhaustive search for the best matching is performed.

The value of precision should normally not be supplied by the user. If ccode = TRUE, this value is preset to the highest exponent of 10 that the C code still can handle (usually 9). If ccode = FALSE, the value is preset according to q (usually 15 if q is small), which can sometimes be changed to obtain less severe warning messages.

Value

Normally an object of class pppmatching that contains detailed information about the parameters used and the resulting distance. See pppmatching.object for details. If matching = FALSE, only the numerical value of the distance is returned.

Author(s)

Dominic Schuhmacher <dominic.schuhmacher@mathematik.uni-goettingen.de>
http://www.dominic.schuhmacher.name

References

Bertsekas, D.P. (1992). Auction algorithms for network flow problems: a tutorial introduction. Computational Optimization and Applications 1, 7-66.

Luenberger, D.G. (2003). *Linear and nonlinear programming.* Second edition. Kluwer.

Schuhmacher, D. (2014). *transport: optimal transport in various forms.* R package version 0.6-2 (or later)

Schuhmacher, D. and Xia, A. (2008). A new metric between distributions of point processes. *Advances in Applied Probability* **40**, 651–672

Schuhmacher, D., Vo, B.-T. and Vo, B.-N. (2008). A consistent metric for performance evaluation of multi-object filters. *IEEE Transactions on Signal Processing* **56**, 3447–3457.

See Also

pppmatching.object, matchingdist

Examples

```
# equal cardinalities
set.seed(140627)
X <- runifpoint(500)
Y <- runifpoint(500)
m <- pppdist(X, Y)
m
## Not run:
plot(m)
## End(Not run)

# differing cardinalities
X <- runifpoint(14)
Y <- runifpoint(10)
m1 <- pppdist(X, Y, type="spa")
m2 <- pppdist(X, Y, type="ace")
m3 <- pppdist(X, Y, type="mat", auction=FALSE)
summary(m1)
summary(m2)
summary(m3)
## Not run:
m1$matrix
m2$matrix
m3$matrix
## End(Not run)

# q = Inf
X <- runifpoint(10)
Y <- runifpoint(10)
mx1 <- pppdist(X, Y, q=Inf, matching=FALSE)
mx2 <- pppdist(X, Y, q=Inf, matching=FALSE, ccode=FALSE, approximation=50)
mx3 <- pppdist(X, Y, q=Inf, matching=FALSE, approximation=Inf)
all.equal(mx1,mx2,mx3)
```

```
# sometimes TRUE
all.equal(mx2,mx3)
# very often TRUE
```

---

pppmatching *Create a Point Matching*

Description

Creates an object of class "pppmatching" representing a matching of two planar point patterns (objects of class "ppp").

Usage

```
pppmatching(X, Y, am, type = NULL, cutoff = NULL, q = NULL,
  mdist = NULL)
```

Arguments

X,Y	Two point patterns (objects of class "ppp").
am	An npoints(X) by npoints(Y) matrix with entries ≥ 0 that specifies which points are matched and with what weight; alternatively, an object that can be coerced to this form by as.matrix.
type	A character string giving the type of the matching. One of "spa", "ace" or "mat", or NULL for a generic or unknown matching.
cutoff, q	Numerical values specifying the cutoff value > 0 for interpoint distances and the order $q \in [1, \infty]$ of the average that is applied to them. NULL if not applicable or unknown.
mdist	Numerical value for the distance to be associated with the matching.

Details

The argument am is interpreted as a "generalized adjacency matrix": if the [i,j]-th entry is positive, then the i-th point of X and the j-th point of Y are matched and the value of the entry gives the corresponding weight of the match. For an unweighted matching all the weights should be set to 1.

The remaining arguments are optional and allow to save additional information about the matching. See the help files for pppdist and matchingdist for details on the meaning of these parameters.

Author(s)

Dominic Schuhmacher <dominic.schuhmacher@stat.unibe.ch> http://www.dominic.schuhmacher.name

See Also

pppmatching.object matchingdist

Examples

```
  # a random unweighted complete matching
  X <- runifpoint(10)
  Y <- runifpoint(10)
  am <- r2dtable(1, rep(1,10), rep(1,10))[[1]]
        # generates a random permutation matrix
  m <- pppmatching(X, Y, am)
  summary(m)
  m$matrix
  ## Not run:
    plot(m)

## End(Not run)

  # a random weighted complete matching
  X <- runifpoint(7)
  Y <- runifpoint(7)
  am <- r2dtable(1, rep(10,7), rep(10,7))[[1]]/10
        # generates a random doubly stochastic matrix
  m2 <- pppmatching(X, Y, am)
  summary(m2)
  m2$matrix
  ## Not run:
    # Note: plotting does currently not distinguish
    # between different weights
    plot(m2)

## End(Not run)
```

`pppmatching.object` *Class of Point Matchings*

Description

A class "pppmatching" to represent a matching of two planar point patterns. Optionally includes information about the construction of the matching and its associated distance between the point patterns.

Details

This class represents a (possibly weighted and incomplete) matching between two planar point patterns (objects of class "ppp").

A matching can be thought of as a bipartite weighted graph where the vertices are given by the two point patterns and edges of positive weights are drawn each time a point of the first point pattern is "matched" with a point of the second point pattern.

If m is an object of type pppmatching, it contains the following elements

pp1, pp2	the two point patterns to be matched (vertices)
matrix	a matrix specifying which points are matched and with what weights (edges)
type	(optional) a character string for the type of the matching (one of "spa", "ace" or "mat")

cutoff	(optional) cutoff value for interpoint distances
q	(optional) the order for taking averages of interpoint distances
distance	(optional) the distance associated with the matching

The element matrix is a "generalized adjacency matrix". The numbers of rows and columns match the cardinalities of the first and second point patterns, respectively. The [i,j]-th entry is positive if the i-th point of X and the j-th point of Y are matched (zero otherwise) and its value then gives the corresponding weight of the match. For an unweighted matching all the weights are set to 1.

The optional elements are for saving details about matchings in the context of optimal point matching techniques. type can be one of "spa" (for "subpattern assignment"), "ace" (for "assignment only if cardinalities differ") or "mat" (for "mass transfer"). cutoff is a positive numerical value that specifies the maximal interpoint distance and q is a value in $[1, \infty]$ that gives the order of the average applied to the interpoint distances. See the help files for pppdist and matchingdist for detailed information about these elements.

Objects of class "pppmatching" may be created by the function pppmatching, and are most commonly obtained as output of the function pppdist. There are methods plot, print and summary for this class.

Author(s)

Dominic Schuhmacher <dominic.schuhmacher@stat.unibe.ch> http://www.dominic.schuhmacher.name

See Also

matchingdist pppmatching

Examples

```
  # a random complete unweighted matching
  X <- runifpoint(10)
  Y <- runifpoint(10)
  am <- r2dtable(1, rep(1,10), rep(1,10))[[1]]
        # generates a random permutation matrix
  m <- pppmatching(X, Y, am)
  summary(m)
  m$matrix
  ## Not run:
    plot(m)

## End(Not run)

  # an optimal complete unweighted matching
  m2 <- pppdist(X,Y)
  summary(m2)
  m2$matrix
  ## Not run:
    plot(m2)

## End(Not run)
```

PPversion *Transform a Function into its P-P or Q-Q Version*

Description

Given a function object f containing both the estimated and theoretical versions of a summary function, these operations combine the estimated and theoretical functions into a new function. When plotted, the new function gives either the P-P plot or Q-Q plot of the original f.

Usage

```
PPversion(f, theo = "theo", columns = ".")

QQversion(f, theo = "theo", columns = ".")
```

Arguments

f	The function to be transformed. An object of class "fv".
theo	The name of the column of f that should be treated as the theoretical value of the function.
columns	Character vector, specifying the columns of f to which the transformation will be applied. Either a vector of names of columns of f, or one of the abbreviations recognised by fvnames.

Details

The argument f should be an object of class "fv", containing both empirical estimates $\widehat{f}(r)$ and a theoretical value $f_0(r)$ for a summary function.

The *P–P version* of f is the function $g(x) = \widehat{f}(f_0^{-1}(x))$ where f_0^{-1} is the inverse function of f_0. A plot of $g(x)$ against x is equivalent to a plot of $\widehat{f}(r)$ against $f_0(r)$ for all r. If f is a cumulative distribution function (such as the result of Fest or Gest) then this is a P–P plot, a plot of the observed versus theoretical probabilities for the distribution. The diagonal line $y = x$ corresponds to perfect agreement between observed and theoretical distribution.

The *Q–Q version* of f is the function $h(x) = f_0^{-1}(\widehat{f}(x))$. If f is a cumulative distribution function, a plot of $h(x)$ against x is a Q–Q plot, a plot of the observed versus theoretical quantiles of the distribution. The diagonal line $y = x$ corresponds to perfect agreement between observed and theoretical distribution. Another straight line corresponds to the situation where the observed variable is a linear transformation of the theoretical variable. For a point pattern X, the Q–Q version of Kest(X) is essentially equivalent to Lest(X).

Value

Another object of class "fv".

Author(s)

Tom Lawrence and Adrian Baddeley.

Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

plot.fv

Examples

```
opa <- par(mar=0.1+c(5,5,4,2))
G <- Gest(redwoodfull)
plot(PPversion(G))
plot(QQversion(G))
par(opa)
```

---

ppx *Multidimensional Space-Time Point Pattern*

Description

Creates a multidimensional space-time point pattern with any kind of coordinates and marks.

Usage

```
ppx(data, domain=NULL, coord.type=NULL, simplify=FALSE)
```

Arguments

data	The coordinates and marks of the points. A `data.frame` or `hyperframe`.
domain	Optional. The space-time domain containing the points. An object in some appropriate format, or `NULL`.
coord.type	Character vector specifying how each column of `data` should be interpreted: as a spatial coordinate, a temporal coordinate, a local coordinate or a mark. Entries are partially matched to the values `"spatial"`, `"temporal"`, `"local"` and `"mark"`.
simplify	Logical value indicating whether to simplify the result in special cases. If `simplify=TRUE`, a two-dimensional point pattern will be returned as an object of class `"ppp"`, and a three-dimensional point pattern will be returned as an object of class `"pp3"`. If `simplify=FALSE` (the default) then the result is always an object of class `"ppx"`.

Details

An object of class `"ppx"` represents a marked point pattern in multidimensional space and/or time. There may be any number of spatial coordinates, any number of temporal coordinates, any number of local coordinates, and any number of mark variables. The individual marks may be atomic (numeric values, factor values, etc) or objects of any kind.

The argument `data` should contain the coordinates and marks of the points. It should be a `data.frame` or more generally a `hyperframe` (see `hyperframe`) with one row of data for each point.

Each column of `data` is either a spatial coordinate, a temporal coordinate, a local coordinate, or a mark variable. The argument `coord.type` determines how each column is interpreted. It should be a character vector, of length equal to the number of columns of `data`. It should contain strings that partially match the values `"spatial"`, `"temporal"`, `"local"` and `"mark"`. (The first letters will be sufficient.)

By default (if `coord.type` is missing or `NULL`), columns of numerical data are assumed to represent spatial coordinates, while other columns are assumed to be marks.

Value

Usually an object of class "ppx". If simplify=TRUE the result may be an object of class "ppp" or "pp3".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

pp3, print.ppx

Examples

```
df <- data.frame(x=runif(4),y=runif(4),t=runif(4),
                 age=rep(c("old", "new"), 2),
                 size=runif(4))
X <- ppx(data=df, coord.type=c("s","s","t","m","m"))
X

val <- 20 * runif(4)
E <- lapply(val, function(s) { rpoispp(s) })
hf <- hyperframe(t=val, e=as.listof(E))
Z <- ppx(data=hf, domain=c(0,1))
Z
```

---

predict.dppm — *Prediction from a Fitted Determinantal Point Process Model*

Description

Given a fitted determinantal point process model, these functions compute the fitted intensity.

Usage

```
  ## S3 method for class 'dppm'
fitted(object, ...)

  ## S3 method for class 'dppm'
predict(object, ...)
```

Arguments

object	Fitted determinantal point process model. An object of class "dppm".
...	Arguments passed to fitted.ppm or predict.ppm respectively.

Details

These functions are methods for the generic functions fitted and predict. The argument object should be a determinantal point process model (object of class "dppm") obtained using the function dppm.

The *intensity* of the fitted model is computed, using fitted.ppm or predict.ppm respectively.

Value

The value of `fitted.dppm` is a numeric vector giving the fitted values at the quadrature points.

The value of `predict.dppm` is usually a pixel image (object of class `"im"`), but see `predict.ppm` for details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`dppm`, `plot.dppm`, `fitted.ppm`, `predict.ppm`

Examples

```
fit <- dppm(swedishpines ~ x + y, dppGauss())
predict(fit)
```

`predict.kppm` *Prediction from a Fitted Cluster Point Process Model*

Description

Given a fitted cluster point process model, these functions compute the fitted intensity.

Usage

```
## S3 method for class 'kppm'
fitted(object, ...)

## S3 method for class 'kppm'
predict(object, ...)
```

Arguments

`object`	Fitted cluster point process model. An object of class `"kppm"`.
`...`	Arguments passed to `fitted.ppm` or `predict.ppm` respectively.

Details

These functions are methods for the generic functions `fitted` and `predict`. The argument `object` should be a cluster point process model (object of class `"kppm"`) obtained using the function `kppm`.

The *intensity* of the fitted model is computed, using `fitted.ppm` or `predict.ppm` respectively.

Value

The value of `fitted.kppm` is a numeric vector giving the fitted values at the quadrature points.

The value of `predict.kppm` is usually a pixel image (object of class `"im"`), but see `predict.ppm` for details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

kppm, plot.kppm, vcov.kppm, fitted.ppm, predict.ppm

Examples

```
data(redwood)
fit <- kppm(redwood ~ x, "Thomas")
predict(fit)
```

---

predict.lppm *Predict Point Process Model on Linear Network*

Description

Given a fitted point process model on a linear network, compute the fitted intensity or conditional intensity of the model.

Usage

```
## S3 method for class 'lppm'
predict(object, ...,
                 type = "trend", locations = NULL, new.coef=NULL)
```

Arguments

object	The fitted model. An object of class "lppm", see lppm.
type	Type of values to be computed. Either "trend", "cif" or "se".
locations	Optional. Locations at which predictions should be computed. Either a data frame with two columns of coordinates, or a binary image mask.
new.coef	Optional. Numeric vector of model coefficients, to be used instead of the fitted coefficients coef(object) when calculating the prediction.
...	Optional arguments passed to as.mask to determine the pixel resolution (if locations is missing).

Details

This function computes the fitted poin process intensity, fitted conditional intensity, or standard error of the fitted intensity, for a point process model on a linear network. It is a method for the generic predict for the class "lppm".

The argument object should be an object of class "lppm" (produced by lppm) representing a point process model on a linear network.

Predicted values are computed at the locations given by the argument locations. If this argument is missing, then predicted values are computed at a fine grid of points on the linear network.

- If `locations` is missing or `NULL` (the default), the return value is a pixel image (object of class `"linim"` which inherits class `"im"`) corresponding to a discretisation of the linear network, with numeric pixel values giving the predicted values at each location on the linear network.
- If `locations` is a data frame, the result is a numeric vector of predicted values at the locations specified by the data frame.
- If `locations` is a binary mask, the result is a pixel image with predicted values computed at the pixels of the mask.

Value

A pixel image (object of class `"linim"` which inherits class `"im"`) or a numeric vector, depending on the argument `locations`. See Details.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Ang, Q.W. (2010) *Statistical methodology for events on a network.* Master's thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. *Scandinavian Journal of Statistics* **39**, 591–617.

McSwiggan, G., Nair, M.G. and Baddeley, A. (2012) Fitting Poisson point process models to events on a linear network. Manuscript in preparation.

See Also

`lpp`, `linim`

Examples

```
X <- runiflpp(12, simplenet)
fit <- lppm(X ~ x)
v <- predict(fit, type="trend")
plot(v)
```

`predict.mppm` *Prediction for Fitted Multiple Point Process Model*

Description

Given a fitted multiple point process model obtained by `mppm`, evaluate the spatial trend and/or the conditional intensity of the model. By default, predictions are evaluated over a grid of locations, yielding pixel images of the trend and conditional intensity. Alternatively predictions may be evaluated at specified locations with specified values of the covariates.

Usage

```
## S3 method for class 'mppm'
predict(object, ..., newdata = NULL, type = c("trend", "cif"),
              ngrid = 40, locations=NULL, verbose=FALSE)
```

Arguments

object	The fitted model. An object of class "mppm" obtained from mppm.
...	Ignored.
newdata	New values of the covariates, for which the predictions should be computed. If newdata=NULL, predictions are computed for the original values of the covariates, to which the model was fitted. Otherwise newdata should be a hyperframe (see hyperframe) containing columns of covariates as required by the model. If type includes "cif", then newdata must also include a column of spatial point pattern responses, in order to compute the conditional intensity.
type	Type of predicted values required. A character string or vector of character strings. Options are "trend" for the spatial trend (first-order term) and "cif" or "lambda" for the conditional intensity. Alternatively type="all" selects all options.
ngrid	Dimensions of the grid of spatial locations at which prediction will be performed (if locations=NULL). An integer or a pair of integers.
locations	Optional. The locations at which predictions should be performed. A list of point patterns, with one entry for each row of newdata.
verbose	Logical flag indicating whether to print progress reports.

Details

This function computes the spatial trend and the conditional intensity of a fitted multiple spatial point process model. See Baddeley and Turner (2000) and Baddeley et al (2007) for explanation and examples.

Note that by "spatial trend" we mean the (exponentiated) first order potential and not the intensity of the process. [For example if we fit the stationary Strauss process with parameters β and γ, then the spatial trend is constant and equal to β.] The conditional intensity $\lambda(u, X)$ of the fitted model is evaluated at each required spatial location u, with respect to the response point pattern X.

If locations=NULL, then predictions are performed at an ngrid by ngrid grid of locations in the window for each response point pattern. The result will be a hyperframe containing a column of images of the trend (if selected) and a column of images of the conditional intensity (if selected). The result can be plotted.

If locations is given, then it should be a list of point patterns (objects of class "ppp"). Predictions are performed at these points. The result is a hyperframe containing a column of marked point patterns where the locations each point.

Value

A hyperframe with columns named trend and cif.

If locations=NULL, the entries of the hyperframe are pixel images.

If locations is not null, the entries are marked point patterns constructed by attaching the predicted values to the locations point patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A. and Turner, R. Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** (2000) 283–322.

Baddeley, A., Bischof, L., Sintorn, I.-M., Haggarty, S., Bell, M. and Turner, R. Analysis of a designed experiment where the response is a spatial point pattern. In preparation.

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

See Also

mppm, fitted.mppm, hyperframe

Examples

```
h <- hyperframe(Bugs=waterstriders)
fit <- mppm(Bugs ~ x, data=h, interaction=Strauss(7))
# prediction on a grid
p <- predict(fit)
plot(p$trend)
# prediction at specified locations
loc <- with(h, runifpoint(20, Window(Bugs)))
p2 <- predict(fit, locations=loc)
plot(p2$trend)
```

---

predict.ppm *Prediction from a Fitted Point Process Model*

Description

Given a fitted point process model obtained by ppm, evaluate the spatial trend or the conditional intensity of the model at new locations.

Usage

```
## S3 method for class 'ppm'
predict(object, window=NULL, ngrid=NULL, locations=NULL,
   covariates=NULL,
   type=c("trend", "cif", "intensity", "count"),
   se=FALSE,
   interval=c("none", "confidence", "prediction"),
   level = 0.95,
   X=data.ppm(object), correction,
   ...,
   dimyx=NULL, eps=NULL,
   new.coef=NULL, check=TRUE, repair=TRUE)
```

Arguments

`object`	A fitted point process model, typically obtained from the model-fitting algorithm `ppm`. An object of class "ppm" (see `ppm.object`).
`window`	Optional. A window (object of class "owin") *delimiting* the locations where predictions should be computed. Defaults to the window of the original data used to fit the model `object`.
`ngrid`	Optional. Dimensions of a rectangular grid of locations inside `window` where the predictions should be computed. An integer, or an integer vector of length 2, specifying the number of grid points in the y and x directions. (Incompatible with `locations`. Equivalent to `dimyx`.)
`locations`	Optional. Data giving the exact x, y coordinates (and marks, if required) of locations at which predictions should be computed. Either a point pattern, or a data frame with columns named `x` and `y`, or a binary image mask, or a pixel image. (Incompatible with `ngrid`, `dimyx` and `eps`).
`covariates`	Values of external covariates required by the model. Either a data frame or a list of images. See Details.
`type`	Character string. Indicates which property of the fitted model should be predicted. Options are "trend" for the spatial trend, "cif" or "lambda" for the conditional intensity, "intensity" for the intensity, and "count" for the total number of points in `window`.
`se`	Logical value indicating whether to calculate standard errors as well.
`interval`	String (partially matched) indicating whether to produce estimates (`interval="none"`, the default) or a confidence interval (`interval="confidence"`) or a prediction interval (`interval="prediction"`).
`level`	Coverage probability for the confidence or prediction interval.
`X`	Optional. A point pattern (object of class "ppp") to be taken as the data point pattern when calculating the conditional intensity. The default is to use the original data to which the model was fitted.
`correction`	Name of the edge correction to be used in calculating the conditional intensity. Options include "border" and "none". Other options may include "periodic", "isotropic" and "translate" depending on the model. The default correction is the one that was used to fit `object`.
`...`	Ignored.
`dimyx`	Equivalent to `ngrid`.
`eps`	Width and height of pixels in the prediction grid. A numerical value, or numeric vector of length 2.
`new.coef`	Numeric vector of parameter values to replace the fitted model parameters `coef(object)`.
`check`	Logical value indicating whether to check the internal format of `object`. If there is any possibility that this object has been restored from a dump file, or has otherwise lost track of the environment where it was originally computed, set `check=TRUE`.
`repair`	Logical value indicating whether to repair the internal format of `object`, if it is found to be damaged.

Details

This function computes properties of a fitted spatial point process model (object of class "ppm"). For a Poisson point process it can compute the fitted intensity function, or the expected number of points in a region. For a Gibbs point process it can compute the spatial trend (first order potential), conditional intensity, and approximate intensity of the process. Point estimates, standard errors, confidence intervals and prediction intervals are available.

Given a point pattern dataset, we may fit a point process model to the data using the model-fitting algorithm `ppm`. This returns an object of class "ppm" representing the fitted point process model (see `ppm.object`). The parameter estimates in this fitted model can be read off simply by printing the `ppm` object. The spatial trend, conditional intensity and intensity of the fitted model are evaluated using this function `predict.ppm`.

The default action is to create a rectangular grid of points in the observation window of the data point pattern, and evaluate the spatial trend at these locations.

The argument `type` specifies the values that are desired:

If `type="trend"`**:** the "spatial trend" of the fitted model is evaluated at each required spatial location u. See below.

If `type="cif"`**:** the conditional intensity $\lambda(u, X)$ of the fitted model is evaluated at each required spatial location u, with respect to the data point pattern X.

If `type="intensity"`**:** the intensity $\lambda(u)$ of the fitted model is evaluated at each required spatial location u.

If `type="count"`**:** the expected total number of points (or the expected number of points falling in `window`) is evaluated. If `window` is a tessellation, the expected number of points in each tile of the tessellation is evaluated.

The spatial trend, conditional intensity, and intensity are all equivalent if the fitted model is a Poisson point process. However, if the model is not a Poisson process, then they are all different. The "spatial trend" is the (exponentiated) first order potential, and not the intensity of the process. [For example if we fit the stationary Strauss process with parameters β and γ, then the spatial trend is constant and equal to β, while the intensity is a smaller value.]

The default is to compute an estimate of the desired quantity. If `interval="confidence"` or `interval="prediction"`, the estimate is replaced by a confidence interval or prediction interval.

If `se=TRUE`, then a standard error is also calculated, and is returned together with the (point or interval) estimate.

The spatial locations where predictions are required, are determined by the (incompatible) arguments `ngrid` and `locations`.

- If the argument `ngrid` is present, then predictions are performed at a rectangular grid of locations in the window `window`. The result of prediction will be a pixel image or images.
- If `locations` is present, then predictions will be performed at the spatial locations given by this dataset. These may be an arbitrary list of spatial locations, or they may be a rectangular grid. The result of prediction will be either a numeric vector or a pixel image or images.
- If neither `ngrid` nor `locations` is given, then `ngrid` is assumed. The value of `ngrid` defaults to `spatstat.options("npixel")`, which is initialised to 128 when **spatstat** is loaded.

The argument `locations` may be a point pattern, a data frame or a list specifying arbitrary locations; or it may be a binary image mask (an object of class "owin" with type "mask") or a pixel image (object of class "im") specifying (a subset of) a rectangular grid of locations.

- If locations is a point pattern (object of class "ppp"), then prediction will be performed at the points of the point pattern. The result of prediction will be a vector of predicted values, one value for each point. If the model is a marked point process, then locations should be a marked point pattern, with marks of the same kind as the model; prediction will be performed at these marked points. The result of prediction will be a vector of predicted values, one value for each (marked) point.
- If locations is a data frame or list, then it must contain vectors locations$x and locations$y specifying the x, y coordinates of the prediction locations. Additionally, if the model is a marked point process, then locations must also contain a factor locations$marks specifying the marks of the prediction locations. These vectors must have equal length. The result of prediction will be a vector of predicted values, of the same length.
- If locations is a binary image mask, then prediction will be performed at each pixel in this binary image where the pixel value is TRUE (in other words, at each pixel that is inside the window). If the fitted model is an unmarked point process, then the result of prediction will be an image. If the fitted model is a marked point process, then prediction will be performed for each possible value of the mark at each such location, and the result of prediction will be a list of images, one for each mark value.
- If locations is a pixel image (object of class "im"), then prediction will be performed at each pixel in this image where the pixel value is defined (i.e.\ where the pixel value is not NA).

The argument covariates gives the values of any spatial covariates at the prediction locations. If the trend formula in the fitted model involves spatial covariates (other than the Cartesian coordinates x, y) then covariates is required. The format and use of covariates are analogous to those of the argument of the same name in ppm. It is either a data frame or a list of images.

- If covariates is a list of images, then the names of the entries should correspond to the names of covariates in the model formula trend. Each entry in the list must be an image object (of class "im", see im.object). The software will look up the pixel values of each image at the quadrature points.
- If covariates is a data frame, then the ith row of covariates is assumed to contain covariate data for the ith location. When locations is a data frame, this just means that each row of covariates contains the covariate data for the location specified in the corresponding row of locations. When locations is a binary image mask, the row covariates[i,] must correspond to the location x[i],y[i] where x = as.vector(raster.x(locations)) and y = as.vector(raster.y(locations)).

Note that if you only want to use prediction in order to generate a plot of the predicted values, it may be easier to use plot.ppm which calls this function and plots the results.

Value

If total *is given:* a numeric vector or matrix.

If locations *is given and is a data frame:* a vector of predicted values for the spatial locations (and marks, if required) given in locations.

If ngrid *is given, or if* locations *is given and is a binary image mask or a pixel image:* If object is an unmarked point process, the result is a pixel image object (of class "im", see im.object) containing the predictions. If object is a multitype point process, the result is a list of pixel images, containing the predictions for each type at the same grid of locations.

The “predicted values” are either values of the spatial trend (if type="trend"), values of the conditional intensity (if type="cif" or type="lambda"), values of the intensity (if type="intensity") or numbers of points (if type="count").

If se=TRUE, then the result is a list with two entries, the first being the predicted values in the format described above, and the second being the standard errors in the same format.

Warnings

The current implementation invokes predict.glm so that **prediction is wrong** if the trend formula in object involves terms in ns(), bs() or poly(). This is a weakness of predict.glm itself!

Error messages may be very opaque, as they tend to come from deep in the workings of predict.glm. If you are passing the covariates argument and the function crashes, it is advisable to start by checking that all the conditions listed above are satisfied.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Turner, R. Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** (2000) 283–322.

Berman, M. and Turner, T.R. Approximating point process likelihoods with GLIM. *Applied Statistics* **41** (1992) 31–38.

See Also

ppm, ppm.object, plot.ppm, print.ppm, fitted.ppm, spatstat.options

Examples

```
  m <- ppm(cells ~ polynom(x,y,2), Strauss(0.05))
  trend <- predict(m, type="trend")
  ## Not run:
  image(trend)
  points(cells)

## End(Not run)
  cif <- predict(m, type="cif")
  ## Not run:
  persp(cif)

## End(Not run)
  data(japanesepines)
  mj <- ppm(japanesepines ~ harmonic(x,y,2))
  se <- predict(mj, se=TRUE)

  # prediction interval for total number of points
  predict(mj, type="count", interval="p")

  # prediction at arbitrary locations
  predict(mj, locations=data.frame(x=0.3, y=0.4))

  X <- runifpoint(5, Window(japanesepines))
  predict(mj, locations=X, se=TRUE)

  # multitype
```

```
rr <- matrix(0.06, 2, 2)
ma <- ppm(amacrine ~ marks,  MultiStrauss(rr))
Z <- predict(ma)
Z <- predict(ma, type="cif")
predict(ma, locations=data.frame(x=0.8, y=0.5,marks="on"), type="cif")
```

---

predict.rppm   *Make Predictions From a Recursively Partitioned Point Process Model*

Description

Given a model which has been fitted to point pattern data by recursive partitioning, compute the predicted intensity of the model.

Usage

```
## S3 method for class 'rppm'
predict(object, ...)

## S3 method for class 'rppm'
fitted(object, ...)
```

Arguments

`object`	Fitted point process model of class "rppm" produced by the function rppm.
`...`	Optional arguments passed to predict.ppm to specify the locations where prediction is required. (Ignored by fitted.rppm)

Details

These functions are methods for the generic functions fitted and predict. They compute the fitted intensity of a point process model. The argument object should be a fitted point process model of class "rppm" produced by the function rppm.

The fitted method computes the fitted intensity at the original data points, yielding a numeric vector with one entry for each data point.

The predict method computes the fitted intensity at any locations. By default, predictions are calculated at a regular grid of spatial locations, and the result is a pixel image giving the predicted intensity values at these locations.

Alternatively, predictions can be performed at other locations, or a finer grid of locations, or only at certain specified locations, using additional arguments ... which will be interpreted by predict.ppm. Common arguments are ngrid to increase the grid resolution, window to specify the prediction region, and locations to specify the exact locations of predictions. See predict.ppm for details of these arguments.

Predictions are computed by evaluating the explanatory covariates at each desired location, and applying the recursive partitioning rule to each set of covariate values.

Value

The result of `fitted.rppm` is a numeric vector.

The result of `predict.rppm` is a pixel image, a list of pixel images, or a numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`rppm`, `plot.rppm`

Examples

```
fit <- rppm(unmark(gorillas) ~ vegetation, data=gorillas.extra)
plot(predict(fit))
lambdaX <- fitted(fit)
lambdaX[1:5]
# Mondriaan pictures
plot(predict(rppm(redwoodfull ~ x + y)))
points(redwoodfull)
```

`predict.slrm` *Predicted or Fitted Values from Spatial Logistic Regression*

Description

Given a fitted Spatial Logistic Regression model, this function computes the fitted probabilities for each pixel, or the fitted point process intensity, or the values of the linear predictor in each pixel.

Usage

```
 ## S3 method for class 'slrm'
predict(object, ..., type = "intensity",
                         newdata=NULL, window=NULL)
```

Arguments

`object`	a fitted spatial logistic regression model. An object of class `"slrm"`.
`...`	Optional arguments passed to `pixellate` determining the pixel resolution for the discretisation of the point pattern.
`type`	Character string (partially) matching one of `"probabilities"`, `"intensity"` or `"link"`.
`newdata`	Optional. List containing new covariate values for the prediction. See Details.
`window`	Optional. New window in which to predict. An object of class `"owin"`.

Details

This is a method for predict for spatial logistic regression models (objects of class "slrm", usually obtained from the function slrm).

The argument type determines which quantity is computed. If type="intensity"), the value of the point process intensity is computed at each pixel. If type="probabilities") the probability of the presence of a random point in each pixel is computed. If type="link", the value of the linear predictor is computed at each pixel.

If newdata = NULL (the default), the algorithm computes fitted values of the model (based on the data that was originally used to fit the model object).

If newdata is given, the algorithm computes predicted values of the model, using the new values of the covariates provided by newdata. The argument newdata should be a list; names of entries in the list should correspond to variables appearing in the model formula of the object. Each list entry may be a pixel image or a single numeric value.

Value

A pixel image (object of class "im") containing the predicted values for each pixel.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

slrm

Examples

```
X <- rpoispp(42)
fit <- slrm(X ~ x+y)
plot(predict(fit))

data(copper)
X <- copper$SouthPoints
Y <- copper$SouthLines
Z <- distmap(Y)
fitc <- slrm(X ~ Z)
pc <- predict(fitc)

Znew <- distmap(copper$Lines)[copper$SouthWindow]
pcnew <- predict(fitc, newdata=list(Z=Znew))
```

---

print.im *Print Brief Details of an Image*

Description

Prints a very brief description of a pixel image object.

Usage

```
  ## S3 method for class 'im'
print(x, ...)
```

Arguments

`x`	Pixel image (object of class "im").
`...`	Ignored.

Details

A very brief description of the pixel image x is printed.

This is a method for the generic function print.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

print, im.object, summary.im

Examples

```
data(letterR)
U <- as.im(letterR)
U
```

---

print.owin — *Print Brief Details of a Spatial Window*

Description

Prints a very brief description of a window object.

Usage

```
  ## S3 method for class 'owin'
print(x, ..., prefix="window: ")
```

Arguments

`x`	Window (object of class "owin").
`...`	Ignored.
`prefix`	Character string to be printed at the start of the output.

Details

A very brief description of the window x is printed.

This is a method for the generic function print.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

print, print.ppp, summary.owin

Examples

```
owin()  # the unit square

data(demopat)
W <- Window(demopat)
W                    # just says it is polygonal
as.mask(W)           # just says it is a binary image
```

---

print.ppm *Print a Fitted Point Process Model*

Description

Default `print` method for a fitted point process model.

Usage

```
## S3 method for class 'ppm'
print(x,...,
     what=c("all", "model", "trend", "interaction", "se", "errors"))
```

Arguments

x	A fitted point process model, typically obtained from the model-fittingg algorithm ppm. An object of class "ppm".
what	Character vector (partially-matched) indicating what information should be printed.
...	Ignored.

Details

This is the `print` method for the class "ppm". It prints information about the fitted model in a sensible format.

The argument `what` makes it possible to print only some of the information.

If `what` is missing, then by default, standard errors for the estimated coefficients of the model will be printed only if the model is a Poisson point process. To print the standard errors for a non-Poisson model, call `print.ppm` with the argument `what` given explicitly, or reset the default rule by typing `spatstat.options(print.ppm.SE="always")`.

Value

none.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm.object for details of the class "ppm".

ppm for generating these objects.

plot.ppm, predict.ppm

Examples

```
 ## Not run:
 m <- ppm(cells, ~1, Strauss(0.05))
 m

## End(Not run)
```

---

print.ppp *Print Brief Details of a Point Pattern Dataset*

Description

Prints a very brief description of a point pattern dataset.

Usage

```
  ## S3 method for class 'ppp'
print(x, ...)
```

Arguments

x	Point pattern (object of class "ppp").
...	Ignored.

Details

A very brief description of the point pattern x is printed.

This is a method for the generic function print.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

print, print.owin, summary.ppp

Examples

```
data(cells)      # plain vanilla point pattern
cells

data(lansing)    # multitype point pattern
lansing

data(longleaf)    # numeric marks
longleaf

data(demopat)     # weird polygonal window
demopat
```

---

print.psp *Print Brief Details of a Line Segment Pattern Dataset*

Description

Prints a very brief description of a line segment pattern dataset.

Usage

```
## S3 method for class 'psp'
print(x, ...)
```

Arguments

x	Line segment pattern (object of class "psp").
...	Ignored.

Details

A very brief description of the line segment pattern x is printed.

This is a method for the generic function print.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

print, print.owin, summary.psp

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
a
```

print.quad *Print a Quadrature Scheme*

Description

print method for a quadrature scheme.

Usage

```
## S3 method for class 'quad'
print(x,...)
```

Arguments

x	A quadrature scheme object, typically obtained from quadscheme. An object of class "quad".
...	Ignored.

Details

This is the print method for the class "quad". It prints simple information about the quadrature scheme.

See quad.object for details of the class "quad".

Value

none.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quadscheme, quad.object, plot.quad, summary.quad

Examples

```
data(cells)
Q <- quadscheme(cells)
Q
```

profilepl	*Fit Models by Profile Maximum Pseudolikelihood or AIC*

Description

Fits point process models by maximising the profile likelihood, profile pseudolikelihood, profile composite likelihood or AIC.

Usage

```
profilepl(s, f, ..., aic=FALSE, rbord=NULL, verbose = TRUE, fast=TRUE)
```

Arguments

`s`	Data frame containing values of the irregular parameters over which the criterion will be computed.
`f`	Function (such as `Strauss`) that generates an interpoint interaction object, given values of the irregular parameters.
`...`	Data passed to `ppm` to fit the model.
`aic`	Logical value indicating whether to find the parameter values which minimise the AIC (`aic=TRUE`) or maximise the profile likelihood (`aic=FALSE`, the default).
`rbord`	Radius for border correction (same for all models). If omitted, this will be computed from the interactions.
`verbose`	Logical value indicating whether to print progress reports.
`fast`	Logical value indicating whether to use a faster, less accurate model-fitting technique when computing the profile pseudolikelihood. See Section on Speed and Accuracy.

Details

The model-fitting function `ppm` fits point process models to point pattern data. However, only the 'regular' parameters of the model can be fitted by `ppm`. The model may also depend on 'irregular' parameters that must be fixed in any call to `ppm`.

This function `profilepl` is a wrapper which finds the values of the irregular parameters that give the best fit. If `aic=FALSE` (the default), the best fit is the model which maximises the likelihood (if the models are Poisson processes) or maximises the pseudolikelihood or logistic likelihood. If `aic=TRUE` then the best fit is the model which minimises the Akaike Information Criterion `AIC.ppm`.

The argument `s` must be a data frame whose columns contain values of the irregular parameters over which the maximisation is to be performed.

An irregular parameter may affect either the interpoint interaction or the spatial trend.

interaction parameters: in a call to `ppm`, the argument `interaction` determines the interaction between points. It is usually a call to a function such as `Strauss`. The arguments of this call are irregular parameters. For example, the interaction radius parameter r of the Strauss process, determined by the argument `r` to the function `Strauss`, is an irregular parameter.

trend parameters: in a call to `ppm`, the spatial trend may depend on covariates, which are supplied by the argument `covariates`. These covariates may be functions written by the user, of the form `function(x,y,...)`, and the extra arguments `...` are irregular parameters.

The argument f determines the interaction for each model to be fitted. It would typically be one of the functions Poisson, AreaInter, BadGey, DiggleGatesStibbard, DiggleGratton, Fiksel, Geyer, Hardcore, LennardJones, OrdThresh, Softcore, Strauss or StraussHard. Alternatively it could be a function written by the user.

Columns of s which match the names of arguments of f will be interpreted as interaction parameters. Other columns will be interpreted as trend parameters.

The data frame s must provide values for each argument of f, except for the optional arguments, which are those arguments of f that have the default value NA.

To find the best fit, each row of s will be taken in turn. Interaction parameters in this row will be passed to f, resulting in an interaction object. Then ppm will be applied to the data ... using this interaction. Any trend parameters will be passed to ppm through the argument covfunargs. This results in a fitted point process model. The value of the log pseudolikelihood or AIC from this model is stored. After all rows of s have been processed in this way, the row giving the maximum value of log pseudolikelihood will be found.

The object returned by profilepl contains the profile pseudolikelihood (or profile AIC) function, the best fitting model, and other data. It can be plotted (yielding a plot of the log pseudolikelihood or AIC values against the irregular parameters) or printed (yielding information about the best fitting values of the irregular parameters).

In general, f may be any function that will return an interaction object (object of class "interact") that can be used in a call to ppm. Each argument of f must be a single value.

Value

An object of class "profilepl". There are methods for plot, print, summary, simulate, as.ppm, fitin and parameters for objects of this class.

The components of the object include

fit	Best-fitting model
param	The data frame s
iopt	Row index of the best-fitting parameters in s

To extract the best fitting model you can also use as.ppm.

Speed and Accuracy

Computation of the profile pseudolikelihood can be time-consuming. We recommend starting with a small experiment in which s contains only a few rows of values. This will indicate roughly the optimal values of the parameters. Then a full calculation using more finely spaced values can identify the exact optimal values.

It is normal that the procedure appears to slow down at the end. During the computation of the profile pseudolikelihood, the model-fitting procedure is accelerated by omitting some calculations that are not needed for computing the pseudolikelihood. When the optimal parameter values have been identified, they are used to fit the final model in its entirety. Fitting the final model can take longer than computing the profile pseudolikelihood.

If fast=TRUE (the default), then additional shortcuts are taken in order to accelerate the computation of the profile log pseudolikelihood. These shortcuts mean that the values of the profile log pseudolikelihood in the result ($prof) may not be equal to the values that would be obtained if the model was fitted normally. Currently this happens only for the area interaction AreaInter. It may be wise to do a small experiment with fast=TRUE and then a definitive calculation with fast=FALSE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

Examples

```
    # one irregular parameter
    rr <- data.frame(r=seq(0.05,0.15, by=0.01))

    ps <- profilepl(rr, Strauss, cells)
    ps
    if(interactive()) plot(ps)

    # two irregular parameters
    rs <- expand.grid(r=seq(0.05,0.15, by=0.01),sat=1:3)

    pg <- profilepl(rs, Geyer, cells)
    pg
    if(interactive()) {
      plot(pg)
      as.ppm(pg)
    }

    # multitype pattern with a common interaction radius
    ## Not run:
     RR <- data.frame(R=seq(0.03,0.05,by=0.01))
     MS <- function(R) { MultiStrauss(radii=diag(c(R,R))) }
     pm <- profilepl(RR, MS, amacrine ~marks)

## End(Not run)
    ## more information
    summary(pg)
```

---

progressreport *Print Progress Reports*

Description

Prints Progress Reports during a loop or iterative calculation.

Usage

```
progressreport(i, n,
                 every = min(100,max(1, ceiling(n/100))),
                 tick = 1,
                 nperline = NULL,
                 charsperline = getOption("width"),
                 style = spatstat.options("progress"),
                 showtime = NULL,
                 state=NULL)
```

Arguments

`i`	Integer. The current iteration number (from 1 to `n`).
`n`	Integer. The (maximum) number of iterations to be computed.
`every`	Optional integer. Iteration number will be printed when `i` is a multiple of `every`.
`tick`	Optional integer. A tick mark or dot will be printed when `i` is a multiple of `tick`.
`nperline`	Optional integer. Number of iterations per line of output.
`charsperline`	Optional integer. The number of characters in a line of output.
`style`	Character string determining the style of display. Options are `"tty"` (the default), `"tk"` and `"txtbar"`. See Details.
`showtime`	Optional. Logical value indicating whether to print the estimated time remaining. Applies only when `style="tty"`.
`state`	Optional. A list containing the internal data.

Details

This is a convenient function for reporting progress during an iterative sequence of calculations or a suite of simulations.

- If `style="tk"` then `tcltk::tkProgressBar` is used to pop-up a new graphics window showing a progress bar. This requires the package **tcltk**. As `i` increases from 1 to `n`, the bar will lengthen. The arguments `every, tick, nperline, showtime` are ignored.
- If `style="txtbar"` then `txtProgressBar` is used to represent progress as a bar made of text characters in the R interpreter window. As `i` increases from 1 to `n`, the bar will lengthen. The arguments `every, tick, nperline, showtime` are ignored.
- If `style="tty"` (the default), then progress reports are printed to the console. This only seems to work well under Linux. As `i` increases from 1 to `n`, the output will be a sequence of dots (one dot for every `tick` iterations), iteration numbers (printed when iteration number is a multiple of `every` or is less than 4), and optionally the estimated time remaining. For example `[etd 1:20:05]` means an estimated time of 1 hour, 20 minutes and 5 seconds until finished.

  The estimated time remaining will be printed only if `style="tty"`, and the argument `state` is given, and either `showtime=TRUE`, or `showtime=NULL` and the iterations are slow (defined as: the estimated time remaining is longer than 3 minutes, or the average time per iteration is longer than 20 seconds).

It is optional, but strongly advisable, to use the argument `state` to store and update the internal data for the progress reports (such as the cumulative time taken for computation) as shown in the last example below. This avoids conflicts with other programs that might be calling `progressreport` at the same time.

Value

If `state` was `NULL`, the result is `NULL`. Otherwise the result is the updated value of `state`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

Examples

```
for(i in 1:40) {
   #
   # code that does something...
   #
   progressreport(i, 40)
}

# saving internal state: *recommended*
sta <- list()
for(i in 1:20) {
   # some code ...
   sta <- progressreport(i, 20, state=sta)
}
```

`project2segment` *Move Point To Nearest Line*

Description

Given a point pattern and a line segment pattern, this function moves each point to the closest location on a line segment.

Usage

```
project2segment(X, Y)
```

Arguments

`X`	A point pattern (object of class `"ppp"`).
`Y`	A line segment pattern (object of class `"psp"`).

Details

For each point `x` in the point pattern `X`, this function finds the closest line segment `y` in the line segment pattern `Y`. It then 'projects' the point `x` onto the line segment `y` by finding the position `z` along `y` which is closest to `x`. This position `z` is returned, along with supplementary information.

Value

A list with the following components. Each component has length equal to the number of points in `X`, and its entries correspond to the points of `X`.

`Xproj`	Point pattern (object of class `"ppp"` containing the projected points.
`mapXY`	Integer vector identifying the nearest segment to each point.
`d`	Numeric vector of distances from each point of `X` to the corresponding projected point.
`tp`	Numeric vector giving the scaled parametric coordinate $0 \leq t_p \leq 1$ of the position of the projected point along the segment.

For example suppose `mapXY[2] = 5` and `tp[2] = 0.33`. Then `Y[5]` is the line segment lying closest to `X[2]`. The projection of the point `X[2]` onto the segment `Y[5]` is the point `Xproj[2]`, which lies one-third of the way between the first and second endpoints of the line segment `Y[5]`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

nearestsegment for a faster way to determine which segment is closest to each point.

Examples

```
X <- rstrat(square(1), 5)
Y <- as.psp(matrix(runif(20), 5, 4), window=owin())
plot(Y, lwd=3, col="green")
plot(X, add=TRUE, col="red", pch=16)
v <- project2segment(X,Y)
Xproj <- v$Xproj
plot(Xproj, add=TRUE, pch=16)
arrows(X$x, X$y, Xproj$x, Xproj$y, angle=10, length=0.15, col="red")
```

---

project2set *Find Nearest Point in a Region*

Description

For each data point in a point pattern X, find the nearest location in a given spatial region W.

Usage

```
project2set(X, W, ...)
```

Arguments

X	Point pattern (object of class "ppp").
W	Window (object of class "owin") or something acceptable to as.owin.
...	Arguments passed to as.mask controlling the pixel resolution.

Details

The window W is first discretised as a binary mask using as.mask.

For each data point X[i] in the point pattern X, the algorithm finds the nearest pixel in W.

The result is a point pattern Y containing these nearest points, that is, Y[i] is the nearest point in W to the point X[i].

Value

A point pattern (object of class "ppp") with the same number of points as X in the window W.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
Rolf Turner <r.turner@auckland.ac.nz>
and Ege Rubak <rubak@math.aau.dk>

See Also

project2segment, nncross

Examples

```
He <- heather$fine[owin(c(2.8, 7.4), c(4.0, 7.8))]
plot(He, main="project2set")
X <- runifpoint(4, erosion(complement.owin(He), 0.2))
points(X, col="red")
Y <- project2set(X, He)
points(Y, col="green")
arrows(X$x, X$y, Y$x, Y$y, angle=15, length=0.2)
```

---

prune.rppm *Prune a Recursively Partitioned Point Process Model*

Description

Given a model which has been fitted to point pattern data by recursive partitioning, apply pruning to reduce the complexity of the partition tree.

Usage

```
## S3 method for class 'rppm'
prune(tree, ...)
```

Arguments

tree	Fitted point process model of class "rppm" produced by the function rppm.
...	Arguments passed to prune.rpart to control the pruning procedure.

Details

This is a method for the generic function prune for the class "rppm". An object of this class is a point process model, fitted to point pattern data by recursive partitioning, by the function rppm.

The recursive partition tree will be pruned using prune.rpart. The result is another object of class "rppm".

Value

Object of class "rppm".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

rppm, plot.rppm, predict.rppm.

Examples

```
# Murchison gold data
mur <- solapply(murchison, rescale, s=1000, unitname="km")
mur$dfault <- distfun(mur$faults)
fit <- rppm(gold ~ dfault + greenstone, data=mur)
fit
prune(fit, cp=0.1)
```

---

pseudoR2 *Calculate Pseudo-R-Squared for Point Process Model*

Description

Given a fitted point process model, calculate the pseudo-R-squared value, which measures the fraction of variation in the data that is explained by the model.

Usage

```
pseudoR2(object, ...)

## S3 method for class 'ppm'
pseudoR2(object, ...)

## S3 method for class 'lppm'
pseudoR2(object, ...)
```

Arguments

object	Fitted point process model. An object of class "ppm" or "lppm".
...	Additional arguments passed to deviance.ppm or deviance.lppm.

Details

The function pseudoR2 is generic, with methods for fitted point process models of class "ppm" and "lppm".

This function computes McFadden's pseudo-Rsquared

$$R^2 = 1 - \frac{D}{D_0}$$

where D is the deviance of the fitted model object, and D_0 is the deviance of the null model (obtained by refitting object using the trend formula ~1). Deviance is defined as twice the negative log-likelihood or log-pseudolikelihood.

Value

A single numeric value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

deviance.ppm, deviance.lppm.

Examples

```
fit <- ppm(swedishpines ~ x+y)
pseudoR2(fit)
```

---

psib *Sibling Probability of Cluster Point Process*

Description

Computes the sibling probability of a cluster point process model.

Usage

```
psib(object)

## S3 method for class 'kppm'
psib(object)
```

Arguments

object	Fitted cluster point process model (object of class "kppm").

Details

In a Poisson cluster process, two points are called *siblings* if they belong to the same cluster, that is, if they had the same parent point. If two points of the process are separated by a distance r, the probability that they are siblings is $p(r) = 1 - 1/g(r)$ where g is the pair correlation function of the process.

The value $p(0) = 1 - 1/g(0)$ is the probability that, if two points of the process are situated very close to each other, they came from the same cluster. This probability is an index of the strength of clustering, with high values suggesting strong clustering.

This concept was proposed in Baddeley, Rubak and Turner (2015, page 479) and Baddeley (2016).

Value

A single number.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Baddeley, A. (2016) Local composite likelihood for spatial point processes. *Spatial Statistics*, in press.

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

See Also

kppm

Examples

```
fit <- kppm(redwood ~1, "Thomas")
psib(fit)
```

---

psp *Create a Line Segment Pattern*

Description

Creates an object of class "psp" representing a line segment pattern in the two-dimensional plane.

Usage

```
psp(x0,y0, x1, y1, window, marks=NULL,
      check=spatstat.options("checksegments"))
```

Arguments

x0	Vector of x coordinates of first endpoint of each segment
y0	Vector of y coordinates of first endpoint of each segment
x1	Vector of x coordinates of second endpoint of each segment
y1	Vector of y coordinates of second endpoint of each segment
window	window of observation, an object of class "owin"
marks	(optional) vector or data frame of mark values
check	Logical value indicating whether to check that the line segments lie inside the window.

Details

In the **spatstat** library, a spatial pattern of line segments is described by an object of class "psp". This function creates such objects.

The vectors x0, y0, x1 and y1 must be numeric vectors of equal length. They are interpreted as the cartesian coordinates of the endpoints of the line segments.

A line segment pattern is assumed to have been observed within a specific region of the plane called the observation window. An object of class "psp" representing a point pattern contains information specifying the observation window. This window must always be specified when creating a point pattern dataset; there is intentionally no default action of "guessing" the window dimensions from the data points alone.

The argument window must be an object of class "owin". It is a full description of the window geometry, and could have been obtained from owin or as.owin, or by just extracting the observation window of another dataset, or by manipulating such windows. See owin or the Examples below.

The optional argument marks is given if the line segment pattern is marked, i.e. if each line segment carries additional information. For example, line segments which are classified into two or more different types, or colours, may be regarded as having a mark which identifies which colour they are.

The object marks must be a vector of the same length as x0, or a data frame with number of rows equal to the length of x0. The interpretation is that marks[i] or marks[i,] is the mark attached to the ith line segment. If the marks are real numbers then marks should be a numeric vector, while if the marks takes only a finite number of possible values (e.g. colours or types) then marks should be a factor.

See psp.object for a description of the class "psp".

Users would normally invoke psp to create a line segment pattern, and the function as.psp to convert data in another format into a line segment pattern.

Value

An object of class "psp" describing a line segment pattern in the two-dimensional plane (see psp.object).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

See Also

psp.object, as.psp, owin.object, owin, as.owin, marks.psp

Examples

```
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
m <- data.frame(A=1:10, B=letters[1:10])
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin(), marks=m)
```

---

psp.object *Class of Line Segment Patterns*

Description

A class "psp" to represent a spatial pattern of line segments in the plane. Includes information about the window in which the pattern was observed. Optionally includes marks.

Details

An object of this class represents a two-dimensional pattern of line segments. It specifies

- the locations of the line segments (both endpoints)
- the window in which the pattern was observed
- optionally, a "mark" attached to each line segment (extra information such as a type label).

If X is an object of type psp, it contains the following elements:

ends	data frame with entries x0, y0, x1, y1 giving coordinates of segment endpoints
window	window of observation

	(an object of class owin)
n	number of line segments
marks	optional vector or data frame of marks
markformat	character string specifying the format of the marks; "none", "vector", or "dataframe"

Users are strongly advised not to manipulate these entries directly.

Objects of class "psp" may be created by the function psp and converted from other types of data by the function as.psp. Note that you must always specify the window of observation; there is intentionally no default action of "guessing" the window dimensions from the line segments alone.

Subsets of a line segment pattern may be obtained by the functions [.psp and clip.psp.

Line segment pattern objects can be plotted just by typing plot(X) which invokes the plot method for line segment pattern objects, plot.psp. See plot.psp for further information.

There are also methods for summary and print for line segment patterns. Use summary(X) to see a useful description of the data.

Utilities for line segment patterns include midpoints.psp (to compute the midpoints of each segment), lengths.psp, (to compute the length of each segment), angles.psp, (to compute the angle of orientation of each segment), and distmap.psp to compute the distance map of a line segment pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

psp, as.psp, [.psp

Examples

```
# creating
    a <- psp(runif(20),runif(20),runif(20),runif(20), window=owin())
# converting from other formats
    a <- as.psp(matrix(runif(80), ncol=4), window=owin())
    a <- as.psp(data.frame(x0=runif(20), y0=runif(20),
                            x1=runif(20), y1=runif(20)), window=owin())
# clipping
    w <- owin(c(0.1,0.7), c(0.2, 0.8))
    b <- clip.psp(a, w)
    b <- a[w]
# the last two lines are equivalent.
```

---

psst *Pseudoscore Diagnostic For Fitted Model against General Alternative*

Description

Given a point process model fitted to a point pattern dataset, and any choice of functional summary statistic, this function computes the pseudoscore test statistic of goodness-of-fit for the model.

Usage

```
psst(object, fun, r = NULL, breaks = NULL, ...,
     model=NULL,
     trend = ~1, interaction = Poisson(), rbord = reach(interaction),
     truecoef=NULL, hi.res=NULL, funargs = list(correction="best"),
     verbose=TRUE)
```

Arguments

`object`	Object to be analysed. Either a fitted point process model (object of class "ppm") or a point pattern (object of class "ppp") or quadrature scheme (object of class "quad").
`fun`	Summary function to be applied to each point pattern.
`r`	Optional. Vector of values of the argument r at which the function $S(r)$ should be computed. This argument is usually not specified. There is a sensible default.
`breaks`	Optional alternative to `r` for advanced use.
`...`	Ignored.
`model`	Optional. A fitted point process model (object of class "ppm") to be re-fitted to the data using `update.ppm`, if `object` is a point pattern. Overrides the arguments `trend,interaction,rbord`.
`trend,interaction,rbord`	Optional. Arguments passed to `ppm` to fit a point process model to the data, if `object` is a point pattern. See `ppm` for details.
`truecoef`	Optional. Numeric vector. If present, this will be treated as if it were the true coefficient vector of the point process model, in calculating the diagnostic. Incompatible with `hi.res`.
`hi.res`	Optional. List of parameters passed to `quadscheme`. If this argument is present, the model will be re-fitted at high resolution as specified by these parameters. The coefficients of the resulting fitted model will be taken as the true coefficients. Then the diagnostic will be computed for the default quadrature scheme, but using the high resolution coefficients.
`funargs`	List of additional arguments to be passed to `fun`.
`verbose`	Logical value determining whether to print progress reports during the computation.

Details

Let x be a point pattern dataset consisting of points $x_1, \ldots, x_n$ in a window W. Consider a point process model fitted to x, with conditional intensity $\lambda(u, x)$ at location u. For the purpose of testing goodness-of-fit, we regard the fitted model as the null hypothesis. Given a functional summary statistic S, consider a family of alternative models obtained by exponential tilting of the null model by S. The pseudoscore for the null model is

$$V(r) = \sum_i \Delta S(x_i, x, r) - \int_W \Delta S(u, x, r)\lambda(u, x)\mathrm{d}u$$

where the Δ operator is

$$\Delta S(u, x, r) = S(x \cup \{u\}, r) - S(x \setminus u, r)$$

the difference between the values of S for the point pattern with and without the point u.

According to the Georgii-Nguyen-Zessin formula, $V(r)$ should have mean zero if the model is correct (ignoring the fact that the parameters of the model have been estimated). Hence $V(r)$ can be used as a diagnostic for goodness-of-fit.

This algorithm computes $V(r)$ by direct evaluation of the sum and integral. It is computationally intensive, but it is available for any summary statistic $S(r)$.

The diagnostic $V(r)$ is also called the **pseudoresidual** of S. On the right hand side of the equation for $V(r)$ given above, the sum over points of x is called the **pseudosum** and the integral is called the **pseudocompensator**.

Value

A function value table (object of class `"fv"`), essentially a data frame of function values.

Columns in this data frame include `dat` for the pseudosum, `com` for the compensator and `res` for the pseudoresidual.

There is a plot method for this class. See `fv.object`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Ege Rubak <rubak@math.aau.dk> and Jesper Møller.

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Special cases: `psstA`, `psstG`.

Alternative functions: `Kres`, `Gres`.

Examples

```
data(cells)
fit0 <- ppm(cells, ~1) # uniform Poisson

G0 <- psst(fit0, Gest)
G0
if(interactive()) plot(G0)
```

`psstA` *Pseudoscore Diagnostic For Fitted Model against Area-Interaction Alternative*

Description

Given a point process model fitted to a point pattern dataset, this function computes the pseudoscore diagnostic of goodness-of-fit for the model, against moderately clustered or moderately inhibited alternatives of area-interaction type.

Usage

```
psstA(object, r = NULL, breaks = NULL, ...,
      model = NULL,
      trend = ~1, interaction = Poisson(),
      rbord = reach(interaction), ppmcorrection = "border",
      correction = "all",
      truecoef = NULL, hi.res = NULL,
      nr=spatstat.options("psstA.nr"),
      ngrid=spatstat.options("psstA.ngrid"))
```

Arguments

`object`	Object to be analysed. Either a fitted point process model (object of class "ppm") or a point pattern (object of class "ppp") or quadrature scheme (object of class "quad").
`r`	Optional. Vector of values of the argument r at which the diagnostic should be computed. This argument is usually not specified. There is a sensible default.
`breaks`	This argument is for internal use only.
`...`	Extra arguments passed to `quadscheme` to determine the quadrature scheme, if `object` is a point pattern.
`model`	Optional. A fitted point process model (object of class "ppm") to be re-fitted to the data using `update.ppm`, if `object` is a point pattern. Overrides the arguments `trend,interaction,rbord,ppmcorrection`.
`trend,interaction,rbord`	Optional. Arguments passed to `ppm` to fit a point process model to the data, if `object` is a point pattern. See `ppm` for details.
`ppmcorrection`	Optional. Character string specifying the edge correction for the pseudolikelihood to be used in fitting the point process model. Passed to `ppm`.
`correction`	Optional. Character string specifying which diagnostic quantities will be computed. Options are "all" and "best". The default is to compute all diagnostic quantities.
`truecoef`	Optional. Numeric vector. If present, this will be treated as if it were the true coefficient vector of the point process model, in calculating the diagnostic. Incompatible with `hi.res`.
`hi.res`	Optional. List of parameters passed to `quadscheme`. If this argument is present, the model will be re-fitted at high resolution as specified by these parameters. The coefficients of the resulting fitted model will be taken as the true coefficients. Then the diagnostic will be computed for the default quadrature scheme, but using the high resolution coefficients.
`nr`	Optional. Number of `r` values to be used if `r` is not specified.
`ngrid`	Integer. Number of points in the square grid used to compute the approximate area.

Details

This function computes the pseudoscore test statistic which can be used as a diagnostic for goodness-of-fit of a fitted point process model.

Let x be a point pattern dataset consisting of points $x_1, \ldots, x_n$ in a window W. Consider a point process model fitted to x, with conditional intensity $\lambda(u, x)$ at location u. For the purpose of testing goodness-of-fit, we regard the fitted model as the null hypothesis. The alternative hypothesis is a family of hybrid models obtained by combining the fitted model with the area-interaction process (see AreaInter). The family of alternatives includes models that are slightly more regular than the fitted model, and others that are slightly more clustered than the fitted model.

The pseudoscore, evaluated at the null model, is

$$V(r) = \sum_i A(x_i, x, r) - \int_W A(u, x, r)\lambda(u, x) \mathrm{d}u$$

where

$$A(u, x, r) = B(x \cup \{u\}, r) - B(x \setminus u, r)$$

where $B(x, r)$ is the area of the union of the discs of radius r centred at the points of x (i.e. $B(x, r)$ is the area of the dilation of x by a distance r). Thus $A(u, x, r)$ is the *unclaimed area* associated with u, that is, the area of that part of the disc of radius r centred at the point u that is not covered by any of the discs of radius r centred at points of x.

According to the Georgii-Nguyen-Zessin formula, $V(r)$ should have mean zero if the model is correct (ignoring the fact that the parameters of the model have been estimated). Hence $V(r)$ can be used as a diagnostic for goodness-of-fit.

The diagnostic $V(r)$ is also called the **pseudoresidual** of S. On the right hand side of the equation for $V(r)$ given above, the sum over points of x is called the **pseudosum** and the integral is called the **pseudocompensator**.

Value

A function value table (object of class "fv"), essentially a data frame of function values.

Columns in this data frame include dat for the pseudosum, com for the compensator and res for the pseudoresidual.

There is a plot method for this class. See fv.object.

Warning

This computation can take a **very long time**.

To shorten the computation time, choose smaller values of the arguments nr and ngrid, or reduce the values of their defaults spatstat.options("psstA.nr") and spatstat.options("psstA.ngrid").

Computation time is roughly proportional to nr * npoints * ngrid^2 where npoints is the number of points in the point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Ege Rubak <rubak@math.aau.dk> and Jesper Møller.

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Alternative functions: psstG, psst, Gres, Kres.

Point process models: ppm.

Options: spatstat.options

Examples

```
pso <- spatstat.options(psstA.ngrid=16,psstA.nr=10)
X <- rStrauss(200,0.1,0.05)
plot(psstA(X))
plot(psstA(X, interaction=Strauss(0.05)))
spatstat.options(pso)
```

---

psstG *Pseudoscore Diagnostic For Fitted Model against Saturation Alternative*

Description

Given a point process model fitted to a point pattern dataset, this function computes the pseudoscore diagnostic of goodness-of-fit for the model, against moderately clustered or moderately inhibited alternatives of saturation type.

Usage

```
psstG(object, r = NULL, breaks = NULL, ...,
      model=NULL,
      trend = ~1, interaction = Poisson(), rbord = reach(interaction),
      truecoef = NULL, hi.res = NULL)
```

Arguments

object	Object to be analysed. Either a fitted point process model (object of class "ppm") or a point pattern (object of class "ppp") or quadrature scheme (object of class "quad").
r	Optional. Vector of values of the argument r at which the diagnostic should be computed. This argument is usually not specified. There is a sensible default.
breaks	Optional alternative to r for advanced use.
...	Ignored.
model	Optional. A fitted point process model (object of class "ppm") to be re-fitted to the data using update.ppm, if object is a point pattern. Overrides the arguments trend,interaction,rbord,ppmcorrection.
trend,interaction,rbord	Optional. Arguments passed to ppm to fit a point process model to the data, if object is a point pattern. See ppm for details.
truecoef	Optional. Numeric vector. If present, this will be treated as if it were the true coefficient vector of the point process model, in calculating the diagnostic. Incompatible with hi.res.
hi.res	Optional. List of parameters passed to quadscheme. If this argument is present, the model will be re-fitted at high resolution as specified by these parameters. The coefficients of the resulting fitted model will be taken as the true coefficients. Then the diagnostic will be computed for the default quadrature scheme, but using the high resolution coefficients.

Details

This function computes the pseudoscore test statistic which can be used as a diagnostic for goodness-of-fit of a fitted point process model.

Consider a point process model fitted to x, with conditional intensity $\lambda(u, x)$ at location u. For the purpose of testing goodness-of-fit, we regard the fitted model as the null hypothesis. The alternative hypothesis is a family of hybrid models obtained by combining the fitted model with the Geyer saturation process (see `Geyer`) with saturation parameter 1. The family of alternatives includes models that are more regular than the fitted model, and others that are more clustered than the fitted model.

For any point pattern x, and any $r > 0$, let $S(x, r)$ be the number of points in x whose nearest neighbour (the nearest other point in x) is closer than r units. Then the pseudoscore for the null model is

$$V(r) = \sum_i \Delta S(x_i, x, r) - \int_W \Delta S(u, x, r)\lambda(u, x)\mathrm{d}u$$

where the Δ operator is

$$\Delta S(u, x, r) = S(x \cup \{u\}, r) - S(x \setminus u, r)$$

the difference between the values of S for the point pattern with and without the point u.

According to the Georgii-Nguyen-Zessin formula, $V(r)$ should have mean zero if the model is correct (ignoring the fact that the parameters of the model have been estimated). Hence $V(r)$ can be used as a diagnostic for goodness-of-fit.

The diagnostic $V(r)$ is also called the **pseudoresidual** of S. On the right hand side of the equation for $V(r)$ given above, the sum over points of x is called the **pseudosum** and the integral is called the **pseudocompensator**.

Value

A function value table (object of class `"fv"`), essentially a data frame of function values.

Columns in this data frame include `dat` for the pseudosum, `com` for the compensator and `res` for the pseudoresidual.

There is a plot method for this class. See `fv.object`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Ege Rubak <rubak@math.aau.dk> and Jesper Møller.

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Alternative functions: `psstA`, `psst`, `Kres`, `Gres`.

Examples

```
X <- rStrauss(200,0.1,0.05)
plot(psstG(X))
plot(psstG(X, interaction=Strauss(0.05)))
```

qqplot.ppm *Q-Q Plot of Residuals from Fitted Point Process Model*

Description

Given a point process model fitted to a point pattern, produce a Q-Q plot based on residuals from the model.

Usage

```
qqplot.ppm(fit, nsim=100, expr=NULL, ..., type="raw",
           style="mean", fast=TRUE, verbose=TRUE, plot.it=TRUE,
           dimyx=NULL, nrep=if(fast) 5e4 else 1e5,
           control=update(default.rmhcontrol(fit), nrep=nrep),
           saveall=FALSE,
           monochrome=FALSE,
           limcol=if(monochrome) "black" else "red",
           maxerr=max(100, ceiling(nsim/10)),
           check=TRUE, repair=TRUE, envir.expr)
```

Arguments

`fit`	The fitted point process model, which is to be assessed using the Q-Q plot. An object of class `"ppm"`. Smoothed residuals obtained from this fitted model will provide the "data" quantiles for the Q-Q plot.
`nsim`	The number of simulations from the "reference" point process model.
`expr`	Determines the simulation mechanism which provides the "theoretical" quantiles for the Q-Q plot. See Details.
`...`	Arguments passed to `diagnose.ppm` influencing the computation of residuals.
`type`	String indicating the type of residuals or weights to be used. Current options are `"eem"` for the Stoyan-Grabarnik exponential energy weights, `"raw"` for the raw residuals, `"inverse"` for the inverse-lambda residuals, and `"pearson"` for the Pearson residuals. A partial match is adequate.
`style`	Character string controlling the type of Q-Q plot. Options are `"classical"` and `"mean"`. See Details.
`fast`	Logical flag controlling the speed and accuracy of computation. Use `fast=TRUE` for interactive use and `fast=FALSE` for publication standard plots. See Details.
`verbose`	Logical flag controlling whether the algorithm prints progress reports during long computations.
`plot.it`	Logical flag controlling whether the function produces a plot or simply returns a value (silently).
`dimyx`	Dimensions of the pixel grid on which the smoothed residual field will be calculated. A vector of two integers.
`nrep`	If `control` is absent, then `nrep` gives the number of iterations of the Metropolis-Hastings algorithm that should be used to generate one simulation of the fitted point process.

control	List of parameters controlling the Metropolis-Hastings algorithm rmh which generates each simulated realisation from the model (unless the model is Poisson). This list becomes the argument control of rmh.default. It overrides nrep.
saveall	Logical flag indicating whether to save all the intermediate calculations.
monochrome	Logical flag indicating whether the plot should be in black and white (monochrome=TRUE), or in colour (monochrome=FALSE).
limcol	String. The colour to be used when plotting the 95% limit curves.
maxerr	Maximum number of failures tolerated while generating simulated realisations. See Details.
check	Logical value indicating whether to check the internal format of fit. If there is any possibility that this object has been restored from a dump file, or has otherwise lost track of the environment where it was originally computed, set check=TRUE.
repair	Logical value indicating whether to repair the internal format of fit, if it is found to be damaged.
envir.expr	Optional. An environment in which the expression expr should be evaluated.

Details

This function generates a Q-Q plot of the residuals from a fitted point process model. It is an addendum to the suite of diagnostic plots produced by the function diagnose.ppm, kept separate because it is computationally intensive. The quantiles of the theoretical distribution are estimated by simulation.

In classical statistics, a Q-Q plot of residuals is a useful diagnostic for checking the distributional assumptions. Analogously, in spatial statistics, a Q-Q plot of the (smoothed) residuals from a fitted point process model is a useful way to check the interpoint interaction part of the model (Baddeley et al, 2005). The systematic part of the model (spatial trend, covariate effects, etc) is assessed using other plots made by diagnose.ppm.

The argument fit represents the fitted point process model. It must be an object of class "ppm" (typically produced by the maximum pseudolikelihood fitting algorithm ppm). Residuals will be computed for this fitted model using residuals.ppm, and the residuals will be kernel-smoothed to produce a "residual field". The values of this residual field will provide the "data" quantiles for the Q-Q plot.

The argument expr is not usually specified. It provides a way to modify the "theoretical" or "reference" quantiles for the Q-Q plot.

In normal usage we set expr=NULL. The default is to generate nsim simulated realisations of the fitted model fit, re-fit this model to each of the simulated patterns, evaluate the residuals from these fitted models, and use the kernel-smoothed residual field from these fitted models as a sample from the reference distribution for the Q-Q plot.

In advanced use, expr may be an expression. It will be re-evaluated nsim times, and should include random computations so that the results are not identical each time. The result of evaluating expr should be either a point pattern (object of class "ppp") or a fitted point process model (object of class "ppm"). If the value is a point pattern, then the original fitted model fit will be fitted to this new point pattern using update.ppm, to yield another fitted model. Smoothed residuals obtained from these nsim fitted models will yield the "theoretical" quantiles for the Q-Q plot.

Alternatively expr can be a list of point patterns, or an envelope object that contains a list of point patterns (typically generated by calling envelope with savepatterns=TRUE). These point patterns will be used as the simulated patterns.

Simulation is performed (if `expr=NULL`) using the Metropolis-Hastings algorithm `rmh`. Each simulated realisation is the result of running the Metropolis-Hastings algorithm from an independent random starting state each time. The iterative and termination behaviour of the Metropolis-Hastings algorithm are governed by the argument `control`. See `rmhcontrol` for information about this argument. As a shortcut, the argument `nrep` determines the number of Metropolis-Hastings iterations used to generate each simulated realisation, if `control` is absent.

By default, simulations are generated in an expanded window. Use the argument `control` to change this, as explained in the section on *Warning messages*.

The argument `type` selects the type of residual or weight that will be computed. For options, see `diagnose.ppm`.

The argument `style` determines the type of Q-Q plot. It is highly recommended to use the default, `style="mean"`.

`style="classical"` The quantiles of the residual field for the data (on the y axis) are plotted against the quantiles of the **pooled** simulations (on the x axis). This plot is biased, and therefore difficult to interpret, because of strong autocorrelations in the residual field and the large differences in sample size.

`style="mean"` The order statistics of the residual field for the data are plotted against the sample means, over the `nsim` simulations, of the corresponding order statistics of the residual field for the simulated datasets. Dotted lines show the 2.5 and 97.5 percentiles, over the `nsim` simulations, of each order statistic.

The argument `fast` is a simple way to control the accuracy and speed of computation. If `fast=FALSE`, the residual field is computed on a fine grid of pixels (by default 100 by 100 pixels, see below) and the Q-Q plot is based on the complete set of order statistics (usually 10,000 quantiles). If `fast=TRUE`, the residual field is computed on a coarse grid (at most 40 by 40 pixels) and the Q-Q plot is based on the *percentiles* only. This is about 7 times faster. It is recommended to use `fast=TRUE` for interactive data analysis and `fast=FALSE` for definitive plots for publication.

The argument `dimyx` gives full control over the resolution of the pixel grid used to calculate the smoothed residuals. Its interpretation is the same as the argument `dimyx` to the function `as.mask`. Note that `dimyx[1]` is the number of pixels in the y direction, and `dimyx[2]` is the number in the x direction. If `dimyx` is not present, then the default pixel grid dimensions are controlled by `spatstat.options("npixel")`.

Since the computation is so time-consuming, `qqplot.ppm` returns a list containing all the data necessary to re-display the Q-Q plot. It is advisable to assign the result of `qqplot.ppm` to something (or use `.Last.value` if you forgot to.) The return value is an object of class `"qqppm"`. There are methods for `plot.qqppm` and `print.qqppm`. See the Examples.

The argument `saveall` is usually set to `FALSE`. If `saveall=TRUE`, then the intermediate results of calculation for each simulated realisation are saved and returned. The return value includes a 3-dimensional array `sim` containing the smoothed residual field images for each of the `nsim` realisations. When `saveall=TRUE`, the return value is an object of very large size, and should not be saved on disk.

Errors may occur during the simulation process, because random data are generated. For example:

- one of the simulated patterns may be empty.
- one of the simulated patterns may cause an error in the code that fits the point process model.
- the user-supplied argument `expr` may have a bug.

Empty point patterns do not cause a problem for the code, but they are reported. Other problems that would lead to a crash are trapped; the offending simulated data are discarded, and the simulation is retried. The argument `maxerr` determines the maximum number of times that such errors will be tolerated (mainly as a safeguard against an infinite loop).

Value

An object of class "qqppm" containing the information needed to reproduce the Q-Q plot. Entries x and y are numeric vectors containing quantiles of the simulations and of the data, respectively.

Side Effects

Produces a Q-Q plot if plot.it is TRUE.

Warning messages

A warning message will be issued if any of the simulations trapped an error (a potential crash).

A warning message will be issued if all, or many, of the simulated point patterns are empty. This usually indicates a problem with the simulation procedure.

The default behaviour of qqplot.ppm is to simulate patterns on an expanded window (specified through the argument control) in order to avoid edge effects. The model's trend is extrapolated over this expanded window. If the trend is strongly inhomogeneous, the extrapolated trend may have very large (or even infinite) values. This can cause the simulation algorithm to produce empty patterns.

The only way to suppress this problem entirely is to prohibit the expansion of the window, by setting the control argument to something like control=list(nrep=1e6, expand=1). Here expand=1 means there will be no expansion. See rmhcontrol for more information about the argument control.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

Stoyan, D. and Grabarnik, P. (1991) Second-order characteristics for stochastic structures connected with Gibbs point processes. *Mathematische Nachrichten*, 151:95–100.

See Also

diagnose.ppm, lurking, residuals.ppm, eem, ppm.object, ppm, rmh, rmhcontrol

Examples

```
    data(cells)

    fit <- ppm(cells, ~1, Poisson())
    diagnose.ppm(fit)  # no suggestion of departure from stationarity
    ## Not run: qqplot.ppm(fit, 80)  # strong evidence of non-Poisson interaction


    ## Not run: 
     diagnose.ppm(fit, type="pearson")
     qqplot.ppm(fit, type="pearson")

## End(Not run)
```

```
############################################
## oops, I need the plot coordinates
mypreciousdata <- .Last.value
## Not run: mypreciousdata <- qqplot.ppm(fit, type="pearson")

plot(mypreciousdata)

#######################################################
# Q-Q plots based on fixed n
# The above QQ plots used simulations from the (fitted) Poisson process.
# But I want to simulate conditional on n, instead of Poisson
# Do this by setting rmhcontrol(p=1)
fixit <- list(p=1)
## Not run: qqplot.ppm(fit, 100, control=fixit)


#######################################################
# Inhomogeneous Poisson data
X <- rpoispp(function(x,y){1000 * exp(-3*x)}, 1000)
plot(X)
# Inhomogeneous Poisson model
fit <- ppm(X, ~x, Poisson())
## Not run: qqplot.ppm(fit, 100)

# conclusion: fitted inhomogeneous Poisson model looks OK

#######################################################
# Advanced use of 'expr' argument
#
# set the initial conditions in Metropolis-Hastings algorithm
#
expr <- expression(rmh(fit, start=list(n.start=42), verbose=FALSE))
## Not run: qqplot.ppm(fit, 100, expr)
```

`quad.object` *Class of Quadrature Schemes*

Description

A class "quad" to represent a quadrature scheme.

Details

A (finite) quadrature scheme is a list of quadrature points u_j and associated weights w_j which is used to approximate an integral by a finite sum:

$$\int f(x)dx \approx \sum_j f(u_j)w_j$$

Given a point pattern dataset, a *Berman-Turner* quadrature scheme is one which includes all these data points, as well as a nonzero number of other ("dummy") points.

These quadrature schemes are used to approximate the pseudolikelihood of a point process, in the method of Baddeley and Turner (2000) (see Berman and Turner (1992)). Accuracy and computation time both increase with the number of points in the quadrature scheme.

An object of class "quad" represents a Berman-Turner quadrature scheme. It can be passed as an argument to the model-fitting function ppm, which requires a quadrature scheme.

An object of this class contains at least the following elements:

data:	an object of class "ppp" giving the locations (and marks) of the data points.
dummy:	an object of class "ppp" giving the locations (and marks) of the dummy points.
w:	vector of nonnegative weights for the quadrature points

Users are strongly advised not to manipulate these entries directly.

The domain of quadrature is specified by `Window(dummy)` while the observation window (if this needs to be specified separately) is taken to be `Window(data)`.

The weights vector `w` may also have an attribute `attr(w, "zeroes")` equivalent to the logical vector `(w == 0)`. If this is absent then all points are known to have positive weights.

To create an object of class "quad", users would typically call the high level function quadscheme. (They are actually created by the low level function quad.)

Entries are extracted from a "quad" object by the functions `x.quad`, `y.quad`, `w.quad` and `marks.quad`, which extract the x coordinates, y coordinates, weights, and marks, respectively. The function `n.quad` returns the total number of quadrature points (dummy plus data).

An object of class "quad" can be converted into an ordinary point pattern by the function `union.quad` which simply takes the union of the data and dummy points.

Quadrature schemes can be plotted using `plot.quad` (a method for the generic `plot`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quadscheme, ppm

---

quad.ppm *Extract Quadrature Scheme Used to Fit a Point Process Model*

Description

Given a fitted point process model, this function extracts the quadrature scheme used to fit the model.

Usage

```
quad.ppm(object, drop=FALSE, clip=FALSE)
```

Arguments

object	fitted point process model (an object of class "ppm" or "kppm" or "lppm").
drop	Logical value determining whether to delete quadrature points that were not used to fit the model.
clip	Logical value determining whether to erode the window, if object was fitted using the border correction. See Details.

Details

An object of class "ppm" represents a point process model that has been fitted to data. It is typically produced by the model-fitting algorithm ppm.

The maximum pseudolikelihood algorithm in ppm approximates the pseudolikelihood integral by a sum over a finite set of quadrature points, which is constructed by augmenting the original data point pattern by a set of "dummy" points. The fitted model object returned by ppm contains complete information about this quadrature scheme. See ppm or ppm.object for further information.

This function quad.ppm extracts the quadrature scheme. A typical use of this function would be to inspect the quadrature scheme (points and weights) to gauge the accuracy of the approximation to the exact pseudolikelihood.

Some quadrature points may not have been used in fitting the model. This happens if the border correction is used, and in other cases (e.g. when the value of a covariate is NA at these points). The argument drop specifies whether these unused quadrature points shall be deleted (drop=TRUE) or retained (drop=FALSE) in the return value.

The quadrature scheme has a *window*, which by default is set to equal the window of the original data. However this window may be larger than the actual domain of integration of the pseudolikelihood or composite likelihood that was used to fit the model. If clip=TRUE then the window of the quadrature scheme is set to the actual domain of integration. This option only has an effect when the model was fitted using the border correction; then the window is obtained by eroding the original data window by the border correction distance.

See ppm.object for a list of all operations that can be performed on objects of class "ppm". See quad.object for a list of all operations that can be performed on objects of class "quad".

This function can also be applied to objects of class "kppm" and "lppm".

Value

A quadrature scheme (object of class "quad").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm.object, quad.object, ppm

Examples

```
fit <- ppm(cells ~1, Strauss(r=0.1))
Q <- quad.ppm(fit)
## Not run: plot(Q)
```

```
npoints(Q$data)
npoints(Q$dummy)
```

quadrat.test *Dispersion Test for Spatial Point Pattern Based on Quadrat Counts*

Description

Performs a test of Complete Spatial Randomness for a given point pattern, based on quadrat counts. Alternatively performs a goodness-of-fit test of a fitted inhomogeneous Poisson model. By default performs chi-squared tests; can also perform Monte Carlo based tests.

Usage

```
quadrat.test(X, ...)

## S3 method for class 'ppp'
quadrat.test(X, nx=5, ny=nx,
                          alternative=c("two.sided", "regular", "clustered"),
                           method=c("Chisq", "MonteCarlo"),
                           conditional=TRUE, CR=1,
                           lambda=NULL,
                           ...,
                           xbreaks=NULL, ybreaks=NULL, tess=NULL,
                           nsim=1999)

## S3 method for class 'ppm'
quadrat.test(X, nx=5, ny=nx,
                          alternative=c("two.sided", "regular", "clustered"),
                           method=c("Chisq", "MonteCarlo"),
                           conditional=TRUE, CR=1,
                           ...,
                           xbreaks=NULL, ybreaks=NULL, tess=NULL,
                           nsim=1999)

## S3 method for class 'quadratcount'
quadrat.test(X,
                          alternative=c("two.sided", "regular", "clustered"),
                          method=c("Chisq", "MonteCarlo"),
                          conditional=TRUE, CR=1,
                          lambda=NULL,
                          ...,
                          nsim=1999)
```

Arguments

X	A point pattern (object of class "ppp") to be subjected to the goodness-of-fit test. Alternatively a fitted point process model (object of class "ppm") to be tested. Alternatively X can be the result of applying quadratcount to a point pattern.
nx,ny	Numbers of quadrats in the x and y directions. Incompatible with xbreaks and ybreaks.

alternative	Character string (partially matched) specifying the alternative hypothesis.
method	Character string (partially matched) specifying the test to use: either method="Chisq" for the chi-squared test (the default), or method="MonteCarlo" for a Monte Carlo test.
conditional	Logical. Should the Monte Carlo test be conducted conditionally upon the observed number of points of the pattern? Ignored if method="Chisq".
CR	Optional. Numerical value of the index λ for the Cressie-Read test statistic.
lambda	Optional. Pixel image (object of class "im") or function (class "funxy") giving the predicted intensity of the point process.
...	Ignored.
xbreaks	Optional. Numeric vector giving the x coordinates of the boundaries of the quadrats. Incompatible with nx.
ybreaks	Optional. Numeric vector giving the y coordinates of the boundaries of the quadrats. Incompatible with ny.
tess	Tessellation (object of class "tess" or something acceptable to as.tess) determining the quadrats. Incompatible with nx, ny, xbreaks, ybreaks.
nsim	The number of simulated samples to generate when method="MonteCarlo".

Details

These functions perform χ^2 tests or Monte Carlo tests of goodness-of-fit for a point process model, based on quadrat counts.

The function quadrat.test is generic, with methods for point patterns (class "ppp"), split point patterns (class "splitppp"), point process models (class "ppm") and quadrat count tables (class "quadratcount").

- if X is a point pattern, we test the null hypothesis that the data pattern is a realisation of Complete Spatial Randomness (the uniform Poisson point process). Marks in the point pattern are ignored. (If lambda is given then the null hypothesis is the Poisson process with intensity lambda.)
- if X is a split point pattern, then for each of the component point patterns (taken separately) we test the null hypotheses of Complete Spatial Randomness. See quadrat.test.splitppp for documentation.
- If X is a fitted point process model, then it should be a Poisson point process model. The data to which this model was fitted are extracted from the model object, and are treated as the data point pattern for the test. We test the null hypothesis that the data pattern is a realisation of the (inhomogeneous) Poisson point process specified by X.

In all cases, the window of observation is divided into tiles, and the number of data points in each tile is counted, as described in quadratcount. The quadrats are rectangular by default, or may be regions of arbitrary shape specified by the argument tess. The expected number of points in each quadrat is also calculated, as determined by CSR (in the first case) or by the fitted model (in the second case). Then the Pearson X^2 statistic

$$X^2 = sum((observed - expected)^2/expected)$$

is computed.

If method="Chisq" then a χ^2 test of goodness-of-fit is performed by comparing the test statistic to the χ^2 distribution with $m - k$ degrees of freedom, where m is the number of quadrats and k is the number of fitted parameters (equal to 1 for quadrat.test.ppp). The default is to compute the *two-sided* p-value, so that the test will be declared significant if X^2 is either very large or very small. One-sided p-values can be obtained by specifying the alternative. An important requirement of the χ^2 test is that the expected counts in each quadrat be greater than 5.

If method="MonteCarlo" then a Monte Carlo test is performed, obviating the need for all expected counts to be at least 5. In the Monte Carlo test, nsim random point patterns are generated from the null hypothesis (either CSR or the fitted point process model). The Pearson X^2 statistic is computed as above. The p-value is determined by comparing the X^2 statistic for the observed point pattern, with the values obtained from the simulations. Again the default is to compute the *two-sided* p-value.

If conditional is TRUE then the simulated samples are generated from the multinomial distribution with the number of "trials" equal to the number of observed points and the vector of probabilities equal to the expected counts divided by the sum of the expected counts. Otherwise the simulated samples are independent Poisson counts, with means equal to the expected counts.

If the argument CR is given, then instead of the Pearson X^2 statistic, the Cressie-Read (1984) power divergence test statistic

$$2nI = \frac{2}{\lambda(\lambda+1)} \sum_i \left[\left(\frac{X_i}{E_i} \right)^\lambda - 1 \right]$$

is computed, where X_i is the ith observed count and E_i is the corresponding expected count, and the exponent λ is equal to CR. The value CR=1 gives the Pearson X^2 statistic; CR=0 gives the likelihood ratio test statistic G^2; CR=-1/2 gives the Freeman-Tukey statistic T^2; CR=-1 gives the modified likelihood ratio test statistic GM^2; and CR=-2 gives Neyman's modified statistic NM^2. In all cases the asymptotic distribution of this test statistic is the same χ^2 distribution as above.

The return value is an object of class "htest". Printing the object gives comprehensible output about the outcome of the test.

The return value also belongs to the special class "quadrat.test". Plotting the object will display the quadrats, annotated by their observed and expected counts and the Pearson residuals. See the examples.

Value

An object of class "htest". See chisq.test for explanation.

The return value is also an object of the special class "quadrattest", and there is a plot method for this class. See the examples.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Cressie, N. and Read, T.R.C. (1984) Multinomial goodness-of-fit tests. *Journal of the Royal Statistical Society, Series B* **46**, 440–464.

See Also

quadrat.test.splitppp, quadratcount, quadrats, quadratresample, chisq.test, cdf.test.

To test a Poisson point process model against a specific alternative, use anova.ppm.

Examples

```
data(simdat)
quadrat.test(simdat)
quadrat.test(simdat, 4, 3)

quadrat.test(simdat, alternative="regular")
quadrat.test(simdat, alternative="clustered")

# Using Monte Carlo p-values
quadrat.test(swedishpines) # Get warning, small expected values.
## Not run:
  quadrat.test(swedishpines, method="M", nsim=4999)
  quadrat.test(swedishpines, method="M", nsim=4999, conditional=FALSE)

## End(Not run)


# quadrat counts
qS <- quadratcount(simdat, 4, 3)
quadrat.test(qS)

# fitted model: inhomogeneous Poisson
fitx <- ppm(simdat, ~x, Poisson())
quadrat.test(fitx)

te <- quadrat.test(simdat, 4)
residuals(te)  # Pearson residuals

plot(te)

plot(simdat, pch="+", cols="green", lwd=2)
plot(te, add=TRUE, col="red", cex=1.4, lty=2, lwd=3)

sublab <- eval(substitute(expression(p[chi^2]==z),
                     list(z=signif(te$p.value,3))))
title(sub=sublab, cex.sub=3)

# quadrats of irregular shape
B <- dirichlet(runifpoint(6, Window(simdat)))
qB <- quadrat.test(simdat, tess=B)
plot(simdat, main="quadrat.test(simdat, tess=B)", pch="+")
plot(qB, add=TRUE, col="red", lwd=2, cex=1.2)
```

---

quadrat.test.mppm — *Chi-Squared Test for Multiple Point Process Model Based on Quadrat Counts*

Description

Performs a chi-squared goodness-of-fit test of a Poisson point process model fitted to multiple point patterns.

Usage

```
## S3 method for class 'mppm'
quadrat.test(X, ...)
```

Arguments

X	An object of class "mppm" representing a point process model fitted to multiple point patterns. It should be a Poisson model.
...	Arguments passed to quadrat.test.ppm which determine the size of the quadrats.

Details

This function performs a χ^2 test of goodness-of-fit for a Poisson point process model, based on quadrat counts. It can also be used to perform a test of Complete Spatial Randomness for a list of point patterns.

The function `quadrat.test` is generic, with methods for point patterns (class "ppp"), point process models (class "ppm") and multiple point process models (class "mppm").

For this function, the argument `X` should be a multiple point process model (object of class "mppm") obtained by fitting a point process model to a list of point patterns using the function `mppm`.

To perform the test, the data point patterns are extracted from `X`. For each point pattern

- the window of observation is divided into rectangular tiles, and the number of data points in each tile is counted, as described in `quadratcount`.
- The expected number of points in each quadrat is calculated, as determined by the fitted model.

Then we perform a single χ^2 test of goodness-of-fit based on these observed and expected counts.

Value

An object of class "htest". Printing the object gives comprehensible output about the outcome of the test. The p-value of the test is stored in the component `p.value`.

The return value also belongs to the special class "quadrat.test". Plotting the object will display, for each window, the position of the quadrats, annotated by their observed and expected counts and the Pearson residuals. See the examples.

The return value also has an attribute "components" which is a list containing the results of χ^2 tests of goodness-of-fit for each individual point pattern.

Testing Complete Spatial Randomness

If the intention is to test Complete Spatial Randomness (CSR) there are two options:

- CSR with the same intensity of points in each point pattern;
- CSR with a different, unrelated intensity of points in each point pattern.

In the first case, suppose `P` is a list of point patterns we want to test. Then fit the multiple model `fit1 <- mppm(P, ~1)` which signifies a Poisson point process model with a constant intensity. Then apply `quadrat.test(fit1)`.

In the second case, fit the model codefit2 <- mppm(P, ~id) which signifies a Poisson point process with a different constant intensity for each point pattern. Then apply `quadrat.test(fit2)`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

See Also

mppm, quadrat.test

Examples

```
H <- hyperframe(X=waterstriders)
# Poisson with constant intensity for all patterns
fit1 <- mppm(X~1, H)
quadrat.test(fit1, nx=2)

# uniform Poisson with different intensity for each pattern
fit2 <- mppm(X ~ id, H)
quadrat.test(fit2, nx=2)
```

---

quadrat.test.splitppp *Dispersion Test of CSR for Split Point Pattern Based on Quadrat Counts*

Description

Performs a test of Complete Spatial Randomness for each of the component patterns in a split point pattern, based on quadrat counts. By default performs chi-squared tests; can also perform Monte Carlo based tests.

Usage

```
## S3 method for class 'splitppp'
quadrat.test(X, ..., df=NULL, df.est=NULL, Xname=NULL)
```

Arguments

X	A split point pattern (object of class "splitppp"), each component of which will be subjected to the goodness-of-fit test.
...	Arguments passed to quadrat.test.ppp.
df,df.est,Xname	Arguments passed to pool.quadrattest.

Details

The function quadrat.test is generic, with methods for point patterns (class "ppp"), split point patterns (class "splitppp") and point process models (class "ppm").

If X is a split point pattern, then for each of the component point patterns (taken separately) we test the null hypotheses of Complete Spatial Randomness, then combine the result into a single test.

The method quadrat.test.ppp is applied to each component point pattern. Then the results are pooled using pool.quadrattest to obtain a single test.

Value

An object of class "quadrattest" which can be printed and plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quadrat.test, quadratcount, quadrats, quadratresample, chisq.test, cdf.test.

To test a Poisson point process model against a specific Poisson alternative, use anova.ppm.

Examples

```
data(humberside)
qH <- quadrat.test(split(humberside), 2, 3)
plot(qH)
qH
```

---

quadratcount *Quadrat counting for a point pattern*

Description

Divides window into quadrats and counts the numbers of points in each quadrat.

Usage

```
  quadratcount(X, ...)

  ## S3 method for class 'ppp'
quadratcount(X, nx=5, ny=nx, ...,
               xbreaks=NULL, ybreaks=NULL, tess=NULL)

  ## S3 method for class 'splitppp'
quadratcount(X, ...)
```

Arguments

X	A point pattern (object of class "ppp") or a split point pattern (object of class "splitppp").
nx,ny	Numbers of rectangular quadrats in the x and y directions. Incompatible with xbreaks and ybreaks.
...	Additional arguments passed to quadratcount.ppp.
xbreaks	Numeric vector giving the x coordinates of the boundaries of the rectangular quadrats. Incompatible with nx.
ybreaks	Numeric vector giving the y coordinates of the boundaries of the rectangular quadrats. Incompatible with ny.
tess	Tessellation (object of class "tess" or something acceptable to as.tess) determining the quadrats. Incompatible with nx,ny,xbreaks,ybreaks.

Details

Quadrat counting is an elementary technique for analysing spatial point patterns. See Diggle (2003).

If X is a point pattern, then by default, the window containing the point pattern X is divided into an nx * ny grid of rectangular tiles or 'quadrats'. (If the window is not a rectangle, then these tiles are intersected with the window.) The number of points of X falling in each quadrat is counted. These numbers are returned as a contingency table.

If xbreaks is given, it should be a numeric vector giving the x coordinates of the quadrat boundaries. If it is not given, it defaults to a sequence of nx+1 values equally spaced over the range of x coordinates in the window Window(X).

Similarly if ybreaks is given, it should be a numeric vector giving the y coordinates of the quadrat boundaries. It defaults to a vector of ny+1 values equally spaced over the range of y coordinates in the window. The lengths of xbreaks and ybreaks may be different.

Alternatively, quadrats of any shape may be used. The argument tess can be a tessellation (object of class "tess") whose tiles will serve as the quadrats.

The algorithm counts the number of points of X falling in each quadrat, and returns these counts as a contingency table.

The return value is a table which can be printed neatly. The return value is also a member of the special class "quadratcount". Plotting the object will display the quadrats, annotated by their counts. See the examples.

To perform a chi-squared test based on the quadrat counts, use quadrat.test.

To calculate an estimate of intensity based on the quadrat counts, use intensity.quadratcount.

To extract the quadrats used in a quadratcount object, use as.tess.

If X is a split point pattern (object of class "splitppp" then quadrat counting will be performed on each of the components point patterns, and the resulting contingency tables will be returned in a list. This list can be printed or plotted.

Marks attached to the points are ignored by quadratcount.ppp. To obtain a separate contingency table for each type of point in a multitype point pattern, first separate the different points using split.ppp, then apply quadratcount.splitppp. See the Examples.

Value

The value of quadratcount.ppp is a contingency table containing the number of points in each quadrat. The table is also an object of the special class "quadratcount" and there is a plot method for this class.

The value of quadratcount.splitppp is a list of such contingency tables, each containing the quadrat counts for one of the component point patterns in X. This list also has the class "solist" which has print and plot methods.

Warning

If Q is the result of quadratcount using rectangular tiles, then as.numeric(Q) extracts the counts **in the wrong order**. To obtain the quadrat counts in the same order as the tiles of the corresponding tessellation would be listed, use as.vector(t(Q)), which works in all cases.

Note

To perform a chi-squared test based on the quadrat counts, use quadrat.test.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 2003.

Stoyan, D. and Stoyan, H. (1994) *Fractals, random shapes and point fields: methods of geometrical statistics.* John Wiley and Sons.

See Also

plot.quadratcount, intensity.quadratcount, quadrats, quadrat.test, tess, hextess, quadratresample, miplot

Examples

```
X <- runifpoint(50)
quadratcount(X)
quadratcount(X, 4, 5)
quadratcount(X, xbreaks=c(0, 0.3, 1), ybreaks=c(0, 0.4, 0.8, 1))
qX <-  quadratcount(X, 4, 5)

# plotting:
plot(X, pch="+")
plot(qX, add=TRUE, col="red", cex=1.5, lty=2)

# irregular window
data(humberside)
plot(humberside)
qH <- quadratcount(humberside, 2, 3)
plot(qH, add=TRUE, col="blue", cex=1.5, lwd=2)

# multitype - split
plot(quadratcount(split(humberside), 2, 3))

# quadrats determined by tessellation:
B <- dirichlet(runifpoint(6))
qX <- quadratcount(X, tess=B)
plot(X, pch="+")
```

```
plot(qX, add=TRUE, col="red", cex=1.5, lty=2)
```

---

quadratresample *Resample a Point Pattern by Resampling Quadrats*

Description

Given a point pattern dataset, create a resampled point pattern by dividing the window into rectangular quadrats and randomly resampling the list of quadrats.

Usage

```
quadratresample(X, nx, ny=nx, ...,
                replace = FALSE, nsamples = 1,
                verbose = (nsamples > 1))
```

Arguments

X	A point pattern dataset (object of class "ppp").
nx,ny	Numbers of quadrats in the x and y directions.
...	Ignored.
replace	Logical value. Specifies whether quadrats should be sampled with or without replacement.
nsamples	Number of randomised point patterns to be generated.
verbose	Logical value indicating whether to print progress reports.

Details

This command implements a very simple bootstrap resampling procedure for spatial point patterns X.

The dataset X must be a point pattern (object of class "ppp") and its observation window must be a rectangle.

The window is first divided into N = nx * ny rectangular tiles (quadrats) of equal size and shape. To generate one resampled point pattern, a random sample of N quadrats is selected from the list of N quadrats, with replacement (if replace=TRUE) or without replacement (if replace=FALSE). The ith quadrat in the original dataset is then replaced by the ith sampled quadrat, after the latter is shifted so that it occupies the correct spatial position. The quadrats are then reconstituted into a point pattern inside the same window as X.

If replace=FALSE, this procedure effectively involves a random permutation of the quadrats. The resulting resampled point pattern has the same number of points as X. If replace=TRUE, the number of points in the resampled point pattern is random.

Value

A point pattern (if nsamples = 1) or a list of point patterns (if nsamples > 1).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quadrats, quadratcount.

See varblock to estimate the variance of a summary statistic by block resampling.

Examples

```
data(bei)
quadratresample(bei, 6, 3)
```

---

quadrats *Divide Region into Quadrats*

Description

Divides window into rectangular quadrats and returns the quadrats as a tessellation.

Usage

```
quadrats(X, nx = 5, ny = nx, xbreaks = NULL, ybreaks = NULL, keepempty=FALSE)
```

Arguments

X	A window (object of class "owin") or anything that can be coerced to a window using as.owin, such as a point pattern.
nx,ny	Numbers of quadrats in the x and y directions. Incompatible with xbreaks and ybreaks.
xbreaks	Numeric vector giving the x coordinates of the boundaries of the quadrats. Incompatible with nx.
ybreaks	Numeric vector giving the y coordinates of the boundaries of the quadrats. Incompatible with ny.
keepempty	Logical value indicating whether to delete or retain empty quadrats. See Details.

Details

If the window X is a rectangle, it is divided into an nx * ny grid of rectangular tiles or 'quadrats'.

If X is not a rectangle, then the bounding rectangle of X is first divided into an nx * ny grid of rectangular tiles, and these tiles are then intersected with the window X.

The resulting tiles are returned as a tessellation (object of class "tess") which can be plotted and used in other analyses.

If xbreaks is given, it should be a numeric vector giving the x coordinates of the quadrat boundaries. If it is not given, it defaults to a sequence of nx+1 values equally spaced over the range of x coordinates in the window Window(X).

Similarly if ybreaks is given, it should be a numeric vector giving the y coordinates of the quadrat boundaries. It defaults to a vector of ny+1 values equally spaced over the range of y coordinates in the window. The lengths of xbreaks and ybreaks may be different.

By default (if keepempty=FALSE), any rectangular tile which does not intersect the window X is ignored, and only the non-empty intersections are treated as quadrats, so the tessellation may consist of fewer than nx * ny tiles. If keepempty=TRUE, empty intersections are retained, and the tessellation always contains exactly nx * ny tiles, some of which may be empty.

Value

A tessellation (object of class "tess") as described under tess.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

tess, quadratcount, quadrat.test, quadratresample

Examples

```
W <- square(10)
Z <- quadrats(W, 4, 5)
plot(Z)

data(letterR)
plot(quadrats(letterR, 5, 7))
```

---

quadscheme *Generate a Quadrature Scheme from a Point Pattern*

Description

Generates a quadrature scheme (an object of class "quad") from point patterns of data and dummy points.

Usage

```
quadscheme(data, dummy, method="grid", ...)
```

Arguments

data	The observed data point pattern. An object of class "ppp" or in a format recognised by as.ppp()
dummy	The pattern of dummy points for the quadrature. An object of class "ppp" or in a format recognised by as.ppp() Defaults to default.dummy(data, ...)
method	The name of the method for calculating quadrature weights: either "grid" or "dirichlet".
...	Parameters of the weighting method (see below) and parameters for constructing the dummy points if necessary.

Details

This is the primary method for producing a quadrature schemes for use by ppm.

The function ppm fits a point process model to an observed point pattern using the Berman-Turner quadrature approximation (Berman and Turner, 1992; Baddeley and Turner, 2000) to the pseudolikelihood of the model. It requires a quadrature scheme consisting of the original data point pattern, an additional pattern of dummy points, and a vector of quadrature weights for all these points. Such quadrature schemes are represented by objects of class "quad". See quad.object for a description of this class.

Quadrature schemes are created by the function quadscheme. The arguments data and dummy specify the data and dummy points, respectively. There is a sensible default for the dummy points (provided by default.dummy). Alternatively the dummy points may be specified arbitrarily and given in any format recognised by as.ppp. There are also functions for creating dummy patterns including corners, gridcentres, stratrand and spokes.

The quadrature region is the region over which we are integrating, and approximating integrals by finite sums. If dummy is a point pattern object (class "ppp") then the quadrature region is taken to be Window(dummy). If dummy is just a list of x, y coordinates then the quadrature region defaults to the observation window of the data pattern, Window(data).

If dummy is missing, then a pattern of dummy points will be generated using default.dummy, taking account of the optional arguments .... By default, the dummy points are arranged in a rectangular grid; recognised arguments include nd (the number of grid points in the horizontal and vertical directions) and eps (the spacing between dummy points). If random=TRUE, a systematic random pattern of dummy points is generated instead. See default.dummy for details.

If method = "grid" then the optional arguments (for ...) are (nd, ntile, eps). The quadrature region (defined above) is divided into an ntile[1] by ntile[2] grid of rectangular tiles. The weight for each quadrature point is the area of a tile divided by the number of quadrature points in that tile.

If method="dirichlet" then the optional arguments are (exact=TRUE, nd, eps). The quadrature points (both data and dummy) are used to construct the Dirichlet tessellation. The quadrature weight of each point is the area of its Dirichlet tile inside the quadrature region. If exact == TRUE then this area is computed exactly using the package deldir; otherwise it is computed approximately by discretisation.

Value

An object of class "quad" describing the quadrature scheme (data points, dummy points, and quadrature weights) suitable as the argument Q of the function ppm() for fitting a point process model.

The quadrature scheme can be inspected using the print and plot methods for objects of class "quad".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Turner, R. Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** (2000) 283–322.

Berman, M. and Turner, T.R. Approximating point process likelihoods with GLIM. *Applied Statistics* **41** (1992) 31–38.

See Also

ppm, as.ppp, quad.object, gridweights, dirichletWeights, corners, gridcentres, stratrand, spokes

Examples

```
  data(simdat)

  # grid weights
  Q <- quadscheme(simdat)
  Q <- quadscheme(simdat, method="grid")
  Q <- quadscheme(simdat, eps=0.5)         # dummy point spacing 0.5 units

  Q <- quadscheme(simdat, nd=50)           # 1 dummy point per tile
  Q <- quadscheme(simdat, ntile=25, nd=50) # 4 dummy points per tile

  # Dirichlet weights
  Q <- quadscheme(simdat, method="dirichlet", exact=FALSE)

  # random dummy pattern
  ## Not run:
  D <- runifpoint(250, Window(simdat))
  Q <- quadscheme(simdat, D, method="dirichlet", exact=FALSE)

## End(Not run)

  # polygonal window
  data(demopat)
  X <- unmark(demopat)
  Q <- quadscheme(X)

  # mask window
  Window(X) <- as.mask(Window(X))
  Q <- quadscheme(X)
```

---

quadscheme.logi   *Generate a Logistic Regression Quadrature Scheme from a Point Pattern*

Description

Generates a logistic regression quadrature scheme (an object of class "logiquad" inheriting from "quad") from point patterns of data and dummy points.

Usage

```
quadscheme.logi(data, dummy, dummytype = "stratrand",
                nd = NULL, mark.repeat = FALSE, ...)
```

Arguments

data	The observed data point pattern. An object of class "ppp" or in a format recognised by as.ppp()
dummy	The pattern of dummy points for the quadrature. An object of class "ppp" or in a format recognised by as.ppp(). If missing a sensible default is generated.
dummytype	The name of the type of dummy points to use when "dummy" is missing. Currently available options are: "stratrand" (default), "binomial", "poisson", "grid" and "transgrid".
nd	Integer, or integer vector of length 2 controlling the intensity of dummy points when "dummy" is missing.
mark.repeat	Repeating the dummy points for each level of a marked data pattern when "dummy" is missing. (See details.)
...	Ignored.

Details

This is the primary method for producing a quadrature schemes for use by ppm when the logistic regression approximation (Baddeley et al. 2013) to the pseudolikelihood of the model is applied (i.e. when method="logi" in ppm).

The function ppm fits a point process model to an observed point pattern. When used with the option method="logi" it requires a quadrature scheme consisting of the original data point pattern and an additional pattern of dummy points. Such quadrature schemes are represented by objects of class "logiquad".

Quadrature schemes are created by the function quadscheme.logi. The arguments data and dummy specify the data and dummy points, respectively. There is a sensible default for the dummy points. Alternatively the dummy points may be specified arbitrarily and given in any format recognised by as.ppp.

The quadrature region is the region over which we are integrating, and approximating integrals by finite sums. If dummy is a point pattern object (class "ppp") then the quadrature region is taken to be Window(dummy). If dummy is just a list of x, y coordinates then the quadrature region defaults to the observation window of the data pattern, Window(data).

If dummy is missing, then a pattern of dummy points will be generated, taking account of the optional arguments dummytype, nd, and mark.repeat.

The currently accepted values for dummytype are:

- "grid" where the frame of the window is divided into a nd * nd or nd[1] * nd[2] regular grid of tiles and the centers constitutes the dummy points.
- "transgrid" where a regular grid as above is translated by a random vector.
- "stratrand" where each point of a regular grid as above is randomly translated within its tile.
- "binomial" where nd * nd or nd[1] * nd[2] points are generated uniformly in the frame of the window. "poisson" where a homogeneous Poisson point process with intensity nd * nd or nd[1] * nd[2] is generated within the frame of observation window.

Then if the window is not rectangular, any dummy points lying outside it are deleted.

If data is a multitype point pattern the dummy points should also be marked (with the same levels of the marks as data). If dummy is missing and the dummy pattern is generated by quadscheme.logi the default behaviour is to attach a uniformly distributed mark (from the levels of the marks) to each dummy point. Alternatively, if `mark.repeat=TRUE` each dummy point is repeated as many times as there are levels of the marks with a distinct mark value attached to it.

Finally, each point (data and dummy) is assigned the weight 1. The weights are never used and only appear to be compatible with the class `"quad"` from which the `"logiquad"` object inherits.

Value

An object of class `"logiquad"` inheriting from `"quad"` describing the quadrature scheme (data points, dummy points, and quadrature weights) suitable as the argument `Q` of the function `ppm()` for fitting a point process model.

The quadrature scheme can be inspected using the `print` and `plot` methods for objects of class `"quad"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> .

References

Baddeley, A., Coeurjolly, J.-F., Rubak, E. and Waagepetersen, R. (2014) Logistic regression for spatial Gibbs point processes. *Biometrika* **101** (2) 377–392.

See Also

`ppm`, `as.ppp`

Examples

```
data(simdat)

Q <- quadscheme.logi(simdat)
```

`quantess` *Quantile Tessellation*

Description

Divide space into tiles which contain equal amounts of stuff.

Usage

```
quantess(M, Z, n, ...)

## S3 method for class 'owin'
quantess(M, Z, n, ..., type=2)

## S3 method for class 'ppp'
quantess(M, Z, n, ..., type=2)

## S3 method for class 'im'
quantess(M, Z, n, ..., type=2)
```

Arguments

- `M` A spatial object (such as a window, point pattern or pixel image) determining the weight or amount of stuff at each location.
- `Z` A spatial covariate (a pixel image or a `function(x,y)`) or one of the strings `"x"` or `"y"` indicating the x or y coordinate. The range of values of `Z` will be broken into `n` bands containing equal amounts of stuff.
- `n` Number of bands. A positive integer.
- `type` Integer specifying the rule for calculating quantiles. Passed to `quantile.default`.
- `...` Additional arguments passed to `quadrats` or `tess` defining another tessellation which should be intersected with the quantile tessellation.

Details

A *quantile tessellation* is a division of space into pieces which contain equal amounts of stuff.

The function `quantess` computes a quantile tessellation and returns the tessellation itself. The function `quantess` is generic, with methods for windows (class `"owin"`), point patterns (`"ppp"`) and pixel images (`"im"`).

The first argument `M` (for mass) specifies the spatial distribution of stuff that is to be divided. If `M` is a window, the *area* of the window is to be divided into `n` equal pieces. If `M` is a point pattern, the *number of points* in the pattern is to be divided into `n` equal parts, as far as possible. If `M` is a pixel image, the pixel values are interpreted as weights, and the *total weight* is to be divided into `n` equal parts.

The second argument `Z` is a spatial covariate. The range of values of `Z` will be divided into `n` bands, each containing the same total weight. That is, we determine the quantiles of `Z` with weights given by `M`.

For convenience, additional arguments `...` can be given, to further subdivide the tiles of the tessellation.

The result of `quantess` is a tessellation of `as.owin(M)` determined by the quantiles of `Z`.

Value

A tessellation (object of class `"tess"`).

Author(s)

Original idea by Ute Hahn. Implemented in `spatstat` by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk> .

See Also

`tess`, `quadrats`, `quantile`, `tilenames`

Examples

```
plot(quantess(letterR, "x", 5))

plot(quantess(bronzefilter, "x", 6))
points(unmark(bronzefilter))
```

```
opa <- par(mar=c(0,0,2,5))
A <- quantess(Window(bei), bei.extra$elev, 4)
plot(A, ribargs=list(las=1))

B <- quantess(bei, bei.extra$elev, 4)
tilenames(B) <- paste(spatstat.utils::ordinal(1:4), "quartile")
plot(B, ribargs=list(las=1))
points(bei, pch=".", cex=2, col="white")
par(opa)
```

---

quantile.density *Quantiles of a Density Estimate*

Description

Given a kernel estimate of a probability density, compute quantiles.

Usage

```
## S3 method for class 'density'
quantile(x, probs = seq(0, 1, 0.25), names = TRUE,
         ..., warn = TRUE)
```

Arguments

x	Object of class "density" computed by a method for density
probs	Numeric vector of probabilities for which the quantiles are required.
names	Logical value indicating whether to attach names (based on probs) to the result.
...	Ignored.
warn	Logical value indicating whether to issue a warning if the density estimate x had to be renormalised because it was computed in a restricted interval.

Details

This function calculates quantiles of the probability distribution whose probability density has been estimated and stored in the object x. The object x must belong to the class "density", and would typically have been obtained from a call to the function density.

The probability density is first normalised so that the total probability is equal to 1. A warning is issued if the density estimate was restricted to an interval (i.e. if x was created by a call to density which included either of the arguments from and to).

Next, the density estimate is numerically integrated to obtain an estimate of the cumulative distribution function $F(x)$. Then for each desired probability p, the algorithm finds the corresponding quantile q.

The quantile q corresponding to probability p satisfies $F(q) = p$ up to the resolution of the grid of values contained in x. The quantile is computed from the right, that is, q is the smallest available value of x such that $F(x) \geq p$.

Value

A numeric vector containing the quantiles.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

quantile, quantile.ewcdf, quantile.im, CDF.

Examples

```
dd <- density(runif(10))
quantile(dd)
```

---

quantile.ewcdf *Quantiles of Weighted Empirical Cumulative Distribution Function*

Description

Compute quantiles of a weighted empirical cumulative distribution function.

Usage

```
  ## S3 method for class 'ewcdf'
quantile(x, probs = seq(0, 1, 0.25),
                 names = TRUE, ...,
                 normalise = TRUE, type=1)
```

Arguments

x	A weighted empirical cumulative distribution function (object of class "ewcdf", produced by ewcdf) for which the quantiles are desired.
probs	probabilities for which the quantiles are desired. A numeric vector of values between 0 and 1.
names	Logical. If TRUE, the resulting vector of quantiles is annotated with names corresponding to probs.
...	Ignored.
normalise	Logical value indicating whether x should first be normalised so that it ranges between 0 and 1.
type	Integer specifying the type of quantile to be calculated, as explained in quantile.default. Only types 1 and 2 are currently implemented.

Details

This is a method for the generic quantile function for the class ewcdf of empirical weighted cumulative distribution functions.

The quantile for a probability p is computed as the right-continuous inverse of the cumulative distribution function x (assuming type=1, the default).

If normalise=TRUE (the default), the weighted cumulative function x is first normalised to have total mass 1 so that it can be interpreted as a cumulative probability distribution function.

Value

Numeric vector of quantiles, of the same length as probs.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> and Kevin Ummel.

See Also

ewcdf, quantile

Examples

```
z <- rnorm(50)
w <- runif(50)
Fun <- ewcdf(z, w)
quantile(Fun, c(0.95,0.99))
```

---

quantile.im *Sample Quantiles of Pixel Image*

Description

Compute the sample quantiles of the pixel values of a given pixel image.

Usage

```
## S3 method for class 'im'
quantile(x, ...)
```

Arguments

x	A pixel image. An object of class "im".
...	Optional arguments passed to quantile.default. They determine the probabilities for which quantiles should be computed. See quantile.default.

Details

This simple function applies the generic quantile operation to the pixel values of the image x.

This function is a convenient way to inspect an image and to obtain summary statistics. See the examples.

Value

A vector of quantiles.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quantile, cut.im, im.object

Examples

```
# artificial image data
Z <- setcov(square(1))

# find the quartiles
quantile(Z)

# find the deciles
quantile(Z, probs=(0:10)/10)
```

---

quasirandom *Quasirandom Patterns*

Description

Generates quasirandom sequences of numbers and quasirandom spatial patterns of points in any dimension.

Usage

```
vdCorput(n, base)

Halton(n, bases = c(2, 3), raw = FALSE, simplify = TRUE)

Hammersley(n, bases = 2, raw = FALSE, simplify = TRUE)
```

Arguments

n	Number of points to generate.
base	A prime number giving the base of the sequence.
bases	Vector of prime numbers giving the bases of the sequences for each coordinate axis.
raw	Logical value indicating whether to return the coordinates as a matrix (raw=TRUE) or as a spatial point pattern (raw=FALSE, the default).
simplify	Argument passed to ppx indicating whether point patterns of dimension 2 or 3 should be returned as objects of class "ppp" or "pp3" respectively (simplify=TRUE, the default) or as objects of class "ppx" (simplify=FALSE).

Details

The function vdCorput generates the quasirandom sequence of Van der Corput (1935) of length n with the given base. These are numbers between 0 and 1 which are in some sense uniformly distributed over the interval.

The function Halton generates the Halton quasirandom sequence of points in d-dimensional space, where d = length(bases). The values of the i-th coordinate of the points are generated using the van der Corput sequence with base equal to bases[i].

The function Hammersley generates the Hammersley set of points in d+1-dimensional space, where d = length(bases). The first d coordinates of the points are generated using the van der Corput sequence with base equal to bases[i]. The d+1-th coordinate is the sequence 1/n, 2/n, ..., 1.

If raw=FALSE (the default) then the Halton and Hammersley sets are interpreted as spatial point patterns of the appropriate dimension. They are returned as objects of class "ppx" (multidimensional point patterns) unless simplify=TRUE and d=2 or d=3 when they are returned as objects of class "ppp" or "pp3". If raw=TRUE, the coordinates are returned as a matrix with n rows and D columns where D is the spatial dimension.

Value

For vdCorput, a numeric vector.

For Halton and Hammersley, an object of class "ppp", "pp3" or "ppx"; or if raw=TRUE, a numeric matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

References

Van der Corput, J. G. (1935) Verteilungsfunktionen. *Proc. Ned. Akad. v. Wetensch.* **38**: 813–821.

Kuipers, L. and Niederreiter, H. (2005) *Uniform distribution of sequences*, Dover Publications.

See Also

rQuasi

Examples

```
vdCorput(10, 2)

plot(Halton(256, c(2,3)))

plot(Hammersley(256, 3))
```

---

rags *Alternating Gibbs Sampler for Multitype Point Processes*

Description

Simulate a realisation of a point process model using the alternating Gibbs sampler.

Usage

```
rags(model, ..., ncycles = 100)
```

Arguments

`model`	Data specifying some kind of point process model.
`...`	Additional arguments passed to other code.
`ncycles`	Number of cycles of the alternating Gibbs sampler that should be performed.

Details

The Alternating Gibbs Sampler for a multitype point process is an iterative simulation procedure. Each step of the sampler updates the pattern of points of a particular type `i`, by drawing a realisation from the conditional distribution of points of type `i` given the points of all other types. Successive steps of the sampler update the points of type 1, then type 2, type 3, and so on.

This is an experimental implementation which currently works only for multitype hard core processes (see `MultiHard`) in which there is no interaction between points of the same type.

The argument `model` should be an object describing a point process model. At the moment, the only permitted format for `model` is of the form `list(beta, hradii)` where `beta` gives the first order trend and `hradii` is the matrix of interaction radii. See `ragsMultiHard` for full details.

Value

A point pattern (object of class `"ppp"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

See Also

`ragsMultiHard`, `ragsAreaInter`

Examples

```
mo <- list(beta=c(30, 20),
           hradii = 0.05 * matrix(c(0,1,1,0), 2, 2))
rags(mo, ncycles=10)
```

`ragsAreaInter` *Alternating Gibbs Sampler for Area-Interaction Process*

Description

Generate a realisation of the area-interaction process using the alternating Gibbs sampler. Applies only when the interaction parameter eta is greater than 1.

Usage

```
ragsAreaInter(beta, eta, r, ...,
              win = NULL, bmax = NULL, periodic = FALSE, ncycles = 100)
```

Arguments

beta	First order trend. A number, a pixel image (object of class "im"), or a function(x,y).
eta	Interaction parameter (canonical form) as described in the help for AreaInter. A number greater than 1.
r	Disc radius in the model. A number greater than 1.
...	Additional arguments for beta if it is a function.
win	Simulation window. An object of class "owin". (Ignored if beta is a pixel image.)
bmax	Optional. The maximum possible value of beta, or a number larger than this.
periodic	Logical value indicating whether to treat opposite sides of the simulation window as being the same, so that points close to one side may interact with points close to the opposite side. Feasible only when the window is a rectangle.
ncycles	Number of cycles of the alternating Gibbs sampler to be performed.

Details

This function generates a simulated realisation of the area-interaction process (see AreaInter) using the alternating Gibbs sampler (see rags).

It exploits a mathematical relationship between the (unmarked) area-interaction process and the two-type hard core process (Baddeley and Van Lieshout, 1995; Widom and Rowlinson, 1970). This relationship only holds when the interaction parameter eta is greater than 1 so that the area-interaction process is clustered.

The parameters beta,eta are the canonical parameters described in the help for AreaInter. The first order trend beta may be a constant, a function, or a pixel image.

The simulation window is determined by beta if it is a pixel image, and otherwise by the argument win (the default is the unit square).

Value

A point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Baddeley, A.J. and Van Lieshout, M.N.M. (1995). Area-interaction point processes. *Annals of the Institute of Statistical Mathematics* **47** (1995) 601–619.

Widom, B. and Rowlinson, J.S. (1970). New model for the study of liquid-vapor phase transitions. *The Journal of Chemical Physics* **52** (1970) 1670–1684.

See Also

rags, ragsMultiHard

AreaInter

Examples

```
plot(ragsAreaInter(100, 2, 0.07, ncycles=15))
```

`ragsMultiHard` *Alternating Gibbs Sampler for Multitype Hard Core Process*

Description

Generate a realisation of the multitype hard core point process using the alternating Gibbs sampler.

Usage

```
ragsMultiHard(beta, hradii, ..., types=NULL, bmax = NULL,
              periodic=FALSE, ncycles = 100)
```

Arguments

`beta`	First order trend. A numeric vector, a pixel image, a function, a list of functions, or a list of pixel images.
`hradii`	Matrix of hard core radii between each pair of types. Diagonal entries should be `0` or `NA`.
`types`	Vector of all possible types for the multitype point pattern.
`...`	Arguments passed to `rmpoispp` when generating random points.
`bmax`	Optional upper bound on `beta`.
`periodic`	Logical value indicating whether to measure distances in the periodic sense, so that opposite sides of the (rectangular) window are treated as identical.
`ncycles`	Number of cycles of the sampler to be performed.

Details

The Alternating Gibbs Sampler for a multitype point process is an iterative simulation procedure. Each step of the sampler updates the pattern of points of a particular type `i`, by drawing a realisation from the conditional distribution of points of type `i` given the points of all other types. Successive steps of the sampler update the points of type 1, then type 2, type 3, and so on.

This is an experimental implementation which currently works only for multitype hard core processes (see `MultiHard`) in which there is no interaction between points of the same type, and for the area-interaction process (see `ragsAreaInter`).

The argument `beta` gives the first order trend for each possible type of point. It may be a single number, a numeric vector, a `function(x,y)`, a pixel image, a list of functions, a `function(x,y,m)`, or a list of pixel images.

The argument `hradii` is the matrix of hard core radii between each pair of possible types of points. Two points of types `i` and `j` respectively are forbidden to lie closer than a distance `hradii[i,j]` apart. The diagonal of this matrix must contain `NA` or `0` values, indicating that there is no hard core constraint applying between points of the same type.

Value

A point pattern (object of class `"ppp"`).

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

See Also

rags, ragsAreaInter

Examples

```
b <- c(30,20)
h <- 0.05 * matrix(c(0,1,1,0), 2, 2)
ragsMultiHard(b, h, ncycles=10)
ragsMultiHard(b, h, ncycles=5, periodic=TRUE)
```

---

ranef.mppm *Extract Random Effects from Point Process Model*

Description

Given a point process model fitted to a list of point patterns, extract the fixed effects of the model. A method for ranef.

Usage

```
## S3 method for class 'mppm'
ranef(object, ...)
```

Arguments

object	A fitted point process model (an object of class "mppm").
...	Ignored.

Details

This is a method for the generic function ranef.

The argument object must be a fitted point process model (object of class "mppm") produced by the fitting algorithm mppm). This represents a point process model that has been fitted to a list of several point pattern datasets. See mppm for information.

This function extracts the coefficients of the random effects of the model.

Value

A data frame, or list of data frames, as described in the help for ranef.lme.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented in **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

See Also

fixef.mppm, coef.mppm

Examples

```
H <- hyperframe(Y = waterstriders)
# Tweak data to exaggerate differences
H$Y[[1]] <- rthin(H$Y[[1]], 0.3)

m1 <- mppm(Y ~ id,  data=H, Strauss(7))
ranef(m1)
m2 <- mppm(Y ~ 1,  random=~1|id, data=H, Strauss(7))
ranef(m2)
```

---

range.fv *Range of Function Values*

Description

Compute the range, maximum, or minimum of the function values in a summary function.

Usage

```
  ## S3 method for class 'fv'
range(..., na.rm = TRUE, finite = na.rm)

  ## S3 method for class 'fv'
max(..., na.rm = TRUE, finite = na.rm)

  ## S3 method for class 'fv'
min(..., na.rm = TRUE, finite = na.rm)
```

Arguments

...	One or more function value tables (objects of class "fv" representing summary functions) or other data.
na.rm	Logical. Whether to ignore NA values.
finite	Logical. Whether to ignore values that are infinite, NaN or NA.

Details

These are methods for the generic range, max and min. They compute the range, maximum, and minimum of the *function* values that would be plotted on the y axis by default.

For more complicated calculations, use with.fv.

Value

Numeric vector of length 2.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

with.fv

Examples

```
G <- Gest(cells)
range(G)
max(G)
min(G)
```

---

raster.x *Cartesian Coordinates for a Pixel Raster*

Description

Return the x and y coordinates of each pixel in a pixel image or binary mask.

Usage

```
raster.x(w, drop=FALSE)
raster.y(w, drop=FALSE)
raster.xy(w, drop=FALSE)
```

Arguments

w	A pixel image (object of class "im") or a mask window (object of class "owin" of type "mask").
drop	Logical. If TRUE, then coordinates of pixels that lie outside the window are removed. If FALSE (the default) then the coordinates of every pixel in the containing rectangle are retained.

Details

The argument w should be either a pixel image (object of class "im") or a mask window (an object of class "owin" of type "mask").

If drop=FALSE (the default), the functions raster.x and raster.y return a matrix of the same dimensions as the pixel image or mask itself, with entries giving the x coordinate (for raster.x) or y coordinate (for raster.y) of each pixel in the pixel grid.

If drop=TRUE, pixels that lie outside the window w (or outside the domain of the image w) are removed, and raster.x and raster.y return numeric vectors containing the coordinates of the pixels that are inside the window w.

The function raster.xy returns a list with components x and y which are numeric vectors of equal length containing the pixel coordinates.

Value

raster.xy returns a list with components x and y which are numeric vectors of equal length containing the pixel coordinates.

If drop=FALSE, raster.x and raster.y return a matrix of the same dimensions as the pixel grid in w, and giving the value of the x (or y) coordinate of each pixel in the raster.

If drop=TRUE, raster.x and raster.y return numeric vectors.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> , Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

owin, as.mask, pixelcentres

Examples

```
u <- owin(c(-1,1),c(-1,1)) # square of side 2
w <- as.mask(u, eps=0.01) # 200 x 200 grid
X <- raster.x(w)
Y <- raster.y(w)
disc <- owin(c(-1,1), c(-1,1), mask=(X^2 + Y^2 <= 1))
## Not run: plot(disc)
# approximation to the unit disc
```

---

rat *Ratio object*

Description

Stores the numerator, denominator, and value of a ratio as a single object.

Usage

```
rat(ratio, numerator, denominator, check = TRUE)
```

Arguments

ratio,numerator,denominator	Three objects belonging to the same class.
check	Logical. Whether to check that the objects are compatible.

Details

The class "rat" is a simple mechanism for keeping track of the numerator and denominator when calculating a ratio. Its main purpose is simply to signal that the object is a ratio.

The function rat creates an object of class "rat" given the numerator, the denominator and the ratio. No calculation is performed; the three objects are simply stored together.

The arguments ratio, numerator, denominator can be objects of any kind. They should belong to the same class. It is assumed that the relationship

$$\text{ratio} = \frac{\text{numerator}}{\text{denominator}}$$

holds in some version of arithmetic. However, no calculation is performed.

By default the algorithm checks whether the three arguments ratio, numerator, denominator are compatible objects, according to compatible.

The result is equivalent to ratio except for the addition of extra information.

Value

An object equivalent to the object ratio except that it also belongs to the class "rat" and has additional attributes numerator and denominator.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

See Also

compatible, pool

---

rCauchy *Simulate Neyman-Scott Point Process with Cauchy cluster kernel*

Description

Generate a random point pattern, a simulated realisation of the Neyman-Scott process with Cauchy cluster kernel.

Usage

```
rCauchy(kappa, scale, mu, win = owin(), thresh = 0.001,
        nsim=1, drop=TRUE,
        saveLambda=FALSE, expand = NULL, ...,
        poisthresh=1e-6, saveparents=TRUE)
```

Arguments

kappa
: Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.

scale
: Scale parameter for cluster kernel. Determines the size of clusters. A positive number, in the same units as the spatial coordinates.

mu
: Mean number of points per cluster (a single positive number) or reference intensity for the cluster points (a function or a pixel image).

win
: Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin.

thresh
: Threshold relative to the cluster kernel value at the origin (parent location) determining when the cluster kernel will be treated as zero for simulation purposes. Will be overridden by argument expand if that is given.

nsim
: Number of simulated realisations to be generated.

drop
: Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

saveLambda
: Logical. If TRUE then the random intensity corresponding to the simulated parent points will also be calculated and saved, and returns as an attribute of the point pattern.

expand
: Numeric. Size of window expansion for generation of parent points. By default determined by calling clusterradius with the numeric threshold value given in thresh.

...
: Passed to clusterfield to control the image resolution when saveLambda=TRUE and to clusterradius when expand is missing or NULL.

poisthresh
: Numerical threshold below which the model will be treated as a Poisson process. See Details.

saveparents
: Logical value indicating whether to save the locations of the parent points as an attribute.

Details

This algorithm generates a realisation of the Neyman-Scott process with Cauchy cluster kernel, inside the window win.

The process is constructed by first generating a Poisson point process of “parent” points with intensity kappa. Then each parent point is replaced by a random cluster of points, the number of points in each cluster being random with a Poisson (mu) distribution, and the points being placed independently and uniformly according to a Cauchy kernel.

In this implementation, parent points are not restricted to lie in the window; the parent process is effectively the uniform Poisson process on the infinite plane.

This model can be fitted to data by the method of minimum contrast, maximum composite likelihood or Palm likelihood using kppm.

The algorithm can also generate spatially inhomogeneous versions of the cluster process:

- The parent points can be spatially inhomogeneous. If the argument kappa is a function(x,y) or a pixel image (object of class "im"), then it is taken as specifying the intensity function of an inhomogeneous Poisson process that generates the parent points.
- The offspring points can be inhomogeneous. If the argument mu is a function(x,y) or a pixel image (object of class "im"), then it is interpreted as the reference density for offspring points, in the sense of Waagepetersen (2006).

When the parents are homogeneous (kappa is a single number) and the offspring are inhomogeneous (mu is a function or pixel image), the model can be fitted to data using kppm.

If the pair correlation function of the model is very close to that of a Poisson process, deviating by less than poisthresh, then the model is approximately a Poisson process, and will be simulated as a Poisson process with intensity kappa * mu, using rpoispp. This avoids computations that would otherwise require huge amounts of memory.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Additionally, some intermediate results of the simulation are returned as attributes of this point pattern (see rNeymanScott). Furthermore, the simulated intensity function is returned as an attribute "Lambda", if saveLambda=TRUE.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.ed

References

Ghorbani, M. (2013) Cauchy cluster process. *Metrika* **76**, 697-706.

Jalilian, A., Guan, Y. and Waagepetersen, R. (2013) Decomposition of variance for spatial Cox processes. *Scandinavian Journal of Statistics* **40**, 119-137.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

rpoispp, rMatClust, rThomas, rVarGamma, rNeymanScott, rGaussPoisson, kppm, clusterfit.

Examples

```
# homogeneous
X <- rCauchy(30, 0.01, 5)
# inhomogeneous
ff <- function(x,y){ exp(2 - 3 * abs(x)) }
Z <- as.im(ff, W= owin())
Y <- rCauchy(50, 0.01, Z)
YY <- rCauchy(ff, 0.01, 5)
```

---

rcell *Simulate Baddeley-Silverman Cell Process*

Description

Generates a random point pattern, a simulated realisation of the Baddeley-Silverman cell process model.

Usage

```
rcell(win=square(1), nx=NULL, ny=nx, ..., dx=NULL, dy=dx,
      N=10, nsim=1, drop=TRUE)
```

Arguments

win	A window. An object of class owin, or data in any format acceptable to as.owin().
nx	Number of columns of cells in the window. Incompatible with dx.
ny	Number of rows of cells in the window. Incompatible with dy.
...	Ignored.
dx	Width of the cells. Incompatible with nx.
dy	Height of the cells. Incompatible with ny.
N	Integer. Distributional parameter: the maximum number of random points in each cell. Passed to rcellnumber.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates a simulated realisation of the "cell process" (Baddeley and Silverman, 1984), a random point process with the same second-order properties as the uniform Poisson process. In particular, the K function of this process is identical to the K function of the uniform Poisson process (aka Complete Spatial Randomness). The same holds for the pair correlation function and all other second-order properties. The cell process is a counterexample to the claim that the K function completely characterises a point pattern.

A cell process is generated by dividing space into equal rectangular tiles. In each tile, a random number of random points is placed. By default, there are either 0, 1 or 10 points, with probabilities $1/10$, $8/9$ and $1/90$ respectively. The points within a tile are independent and uniformly distributed in that tile, and the numbers of points in different tiles are independent random integers.

The tile width is determined either by the number of columns nx or by the horizontal spacing dx. The tile height is determined either by the number of rows ny or by the vertical spacing dy. The cell process is then generated in these tiles. The random numbers of points are generated by rcellnumber.

Some of the resulting random points may lie outside the window win: if they do, they are deleted. The result is a point pattern inside the window win.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A.J. and Silverman, B.W. (1984) A cautionary example on the use of second-order methods for analyzing point patterns. *Biometrics* **40**, 1089-1094.

See Also

rcellnumber, rstrat, rsyst, runifpoint, Kest

Examples

```
X <- rcell(nx=15)
plot(X)
plot(Kest(X))
```

`rcellnumber` *Generate Random Numbers of Points for Cell Process*

Description

Generates random integers for the Baddeley-Silverman counterexample.

Usage

```
rcellnumber(n, N = 10, mu=1)
```

Arguments

`n`	Number of random integers to be generated.
`N`	Distributional parameter: the largest possible value (when `mu <= 1`). An integer greater than 1.
`mu`	Mean of the distribution (equals the variance). Any positive real number.

Details

If `mu = 1` (the default), this function generates random integers which have mean and variance equal to 1, but which do not have a Poisson distribution. The random integers take the values 0, 1 and N with probabilities $1/N$, $(N-2)/(N-1)$ and $1/(N(N-1))$ respectively. See Baddeley and Silverman (1984).

If `mu` is another positive number, the random integers will have mean and variance equal to `mu`. They are obtained by generating the one-dimensional counterpart of the cell process and counting the number of points in the interval from `0` to `mu`. The maximum possible value of each random integer is `N * ceiling(mu)`.

Value

An integer vector of length `n`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<r.turner@auckland.ac.nz>` and Ege Rubak `<rubak@math.aau.dk>`.

References

Baddeley, A.J. and Silverman, B.W. (1984) A cautionary example on the use of second-order methods for analyzing point patterns. *Biometrics* **40**, 1089-1094.

See Also

`rcell`

Examples

```
rcellnumber(30, 3)
```

---

rDGS *Perfect Simulation of the Diggle-Gates-Stibbard Process*

Description

Generate a random pattern of points, a simulated realisation of the Diggle-Gates-Stibbard process, using a perfect simulation algorithm.

Usage

```
rDGS(beta, rho, W = owin(), expand=TRUE, nsim=1, drop=TRUE)
```

Arguments

`beta`	intensity parameter (a positive number).
`rho`	interaction range (a non-negative number).
`W`	window (object of class `"owin"`) in which to generate the random pattern.
`expand`	Logical. If `FALSE`, simulation is performed in the window `W`, which must be rectangular. If `TRUE` (the default), simulation is performed on a larger window, and the result is clipped to the original window `W`. Alternatively `expand` can be an object of class `"rmhexpand"` (see `rmhexpand`) determining the expansion method.
`nsim`	Number of simulated realisations to be generated.
`drop`	Logical. If `nsim=1` and `drop=TRUE` (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates a realisation of the Diggle-Gates-Stibbard point process in the window `W` using a 'perfect simulation' algorithm.

Diggle, Gates and Stibbard (1987) proposed a pairwise interaction point process in which each pair of points separated by a distance d contributes a factor $e(d)$ to the probability density, where

$$e(d) = \sin^2\left(\frac{\pi d}{2\rho}\right)$$

for $d < \rho$, and $e(d)$ is equal to 1 for $d \geq \rho$.

The simulation algorithm used to generate the point pattern is 'dominated coupling from the past' as implemented by Berthelsen and Møller (2002, 2003). This is a 'perfect simulation' or 'exact simulation' algorithm, so called because the output of the algorithm is guaranteed to have the correct probability distribution exactly (unlike the Metropolis-Hastings algorithm used in `rmh`, whose output is only approximately correct).

There is a tiny chance that the algorithm will run out of space before it has terminated. If this occurs, an error message will be generated.

Value

If `nsim = 1`, a point pattern (object of class "ppp"). If `nsim > 1`, a list of point patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, based on original code for the Strauss process by Kasper Klitgaard Berthelsen.

References

Berthelsen, K.K. and Møller, J. (2002) A primer on perfect simulation for spatial point processes. *Bulletin of the Brazilian Mathematical Society* 33, 351-367.

Berthelsen, K.K. and Møller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* 30, 549-564.

Diggle, P.J., Gates, D.J., and Stibbard, A. (1987) A nonparametric estimator for pairwise-interaction point processes. Biometrika **74**, 763 – 770. *Scandinavian Journal of Statistics* **21**, 359–373.

Møller, J. and Waagepetersen, R. (2003). *Statistical Inference and Simulation for Spatial Point Processes.* Chapman and Hall/CRC.

See Also

rmh, DiggleGatesStibbard.

rStrauss, rHardcore, rStraussHard, rDiggleGratton, rPenttinen.

Examples

```
X <- rDGS(50, 0.05)
```

---

rDiggleGratton *Perfect Simulation of the Diggle-Gratton Process*

Description

Generate a random pattern of points, a simulated realisation of the Diggle-Gratton process, using a perfect simulation algorithm.

Usage

```
rDiggleGratton(beta, delta, rho, kappa=1, W = owin(),
               expand=TRUE, nsim=1, drop=TRUE)
```

Arguments

beta	intensity parameter (a positive number).
delta	hard core distance (a non-negative number).
rho	interaction range (a number greater than delta).
kappa	interaction exponent (a non-negative number).

W	window (object of class "owin") in which to generate the random pattern. Currently this must be a rectangular window.
expand	Logical. If FALSE, simulation is performed in the window W, which must be rectangular. If TRUE (the default), simulation is performed on a larger window, and the result is clipped to the original window W. Alternatively expand can be an object of class "rmhexpand" (see rmhexpand) determining the expansion method.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates a realisation of the Diggle-Gratton point process in the window W using a 'perfect simulation' algorithm.

Diggle and Gratton (1984, pages 208-210) introduced the pairwise interaction point process with pair potential $h(t)$ of the form

$$h(t) = \left(\frac{t-\delta}{\rho-\delta}\right)^{\kappa} \qquad \text{if } \delta \leq t \leq \rho$$

with $h(t) = 0$ for $t < \delta$ and $h(t) = 1$ for $t > \rho$. Here δ, ρ and κ are parameters.

Note that we use the symbol κ where Diggle and Gratton (1984) use β, since in **spatstat** we reserve the symbol β for an intensity parameter.

The parameters must all be nonnegative, and must satisfy $\delta \leq \rho$.

The simulation algorithm used to generate the point pattern is 'dominated coupling from the past' as implemented by Berthelsen and Møller (2002, 2003). This is a 'perfect simulation' or 'exact simulation' algorithm, so called because the output of the algorithm is guaranteed to have the correct probability distribution exactly (unlike the Metropolis-Hastings algorithm used in rmh, whose output is only approximately correct).

There is a tiny chance that the algorithm will run out of space before it has terminated. If this occurs, an error message will be generated.

Value

If nsim = 1, a point pattern (object of class "ppp"). If nsim > 1, a list of point patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

based on original code for the Strauss process by Kasper Klitgaard Berthelsen.

References

Berthelsen, K.K. and Møller, J. (2002) A primer on perfect simulation for spatial point processes. *Bulletin of the Brazilian Mathematical Society* 33, 351-367.

Berthelsen, K.K. and Møller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* 30, 549-564.

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

Møller, J. and Waagepetersen, R. (2003). *Statistical Inference and Simulation for Spatial Point Processes.* Chapman and Hall/CRC.

See Also

rmh, DiggleGratton.

rStrauss, rHardcore, rStraussHard, rDGS, rPenttinen.

Examples

```
X <- rDiggleGratton(50, 0.02, 0.07)
```

---

rdpp *Simulation of a Determinantal Point Process*

Description

Generates simulated realisations from a determinantal point process.

Usage

```
rdpp(eig, index, basis = "fourierbasis",
     window = boxx(rep(list(0:1), ncol(index))),
     reject_max = 10000, progress = 0, debug = FALSE, ...)
```

Arguments

eig	vector of values between 0 and 1 specifying the non-zero eigenvalues for the process.
index	data.frame or matrix (or something acceptable to as.matrix) specifying indices of the basis functions.
basis	character string giving the name of the basis.
window	window (of class "owin", "box3" or "boxx") giving the domain of the point process.
reject_max	integer giving the maximal number of trials for rejection sampling.
progress	integer giving the interval for making a progress report. The value zero turns reporting off.
debug	logical value indicating whether debug informationb should be outputted.
...	Ignored.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
index <- expand.grid(-2:2,-2:2)
eig <- exp(-rowSums(index^2))
X <- rdpp(eig, index)
X
## To simulate a det. projection p. p. with the given indices set eig=1:
XX <- rdpp(1, index)
XX
```

---

reach *Interaction Distance of a Point Process*

Description

Computes the interaction distance of a point process.

Usage

```
  reach(x, ...)

  ## S3 method for class 'ppm'
reach(x, ..., epsilon=0)

  ## S3 method for class 'interact'
reach(x, ...)

  ## S3 method for class 'rmhmodel'
reach(x, ...)

  ## S3 method for class 'fii'
reach(x, ..., epsilon)
```

Arguments

`x`	Either a fitted point process model (object of class "ppm"), an interpoint interaction (object of class "interact"), a fitted interpoint interaction (object of class "fii") or a point process model for simulation (object of class "rmhmodel").
`epsilon`	Numerical threshold below which interaction is treated as zero. See details.
`...`	Other arguments are ignored.

Details

The 'interaction distance' or 'interaction range' of a point process model is the smallest distance D such that any two points in the process which are separated by a distance greater than D do not interact with each other.

For example, the interaction range of a Strauss process (see `Strauss`) with parameters β, γ, r is equal to r, unless $\gamma = 1$ in which case the model is Poisson and the interaction range is 0. The interaction range of a Poisson process is zero. The interaction range of the Ord threshold process (see `OrdThresh`) is infinite, since two points *may* interact at any distance apart.

The function `reach(x)` is generic, with methods for the case where `x` is

- a fitted point process model (object of class "ppm", usually obtained from the model-fitting function ppm);
- an interpoint interaction structure (object of class "interact"), created by one of the functions Poisson, Strauss, StraussHard, MultiStrauss, MultiStraussHard, Softcore, DiggleGratton, Pairwise, PairPiece, Geyer, LennardJones, Saturated, OrdThresh or Ord;
- a fitted interpoint interaction (object of class "fii") extracted from a fitted point process model by the command fitin;
- a point process model for simulation (object of class "rmhmodel"), usually obtained from rmhmodel.

When x is an "interact" object, reach(x) returns the maximum possible interaction range for any point process model with interaction structure given by x. For example, reach(Strauss(0.2)) returns 0.2.

When x is a "ppm" object, reach(x) returns the interaction range for the point process model represented by x. For example, a fitted Strauss process model with parameters beta,gamma,r will return either 0 or r, depending on whether the fitted interaction parameter gamma is equal or not equal to 1.

For some point process models, such as the soft core process (see Softcore), the interaction distance is infinite, because the interaction terms are positive for all pairs of points. A practical solution is to compute the distance at which the interaction contribution from a pair of points falls below a threshold epsilon, on the scale of the log conditional intensity. This is done by setting the argument epsilon to a positive value.

Value

The interaction distance, or NA if this cannot be computed from the information given.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, Poisson, Strauss, StraussHard, MultiStrauss, MultiStraussHard, Softcore, DiggleGratton, Pairwise, PairPiece, Geyer, LennardJones, Saturated, OrdThresh, Ord, rmhmodel

Examples

```
reach(Poisson())
# returns 0

reach(Strauss(r=7))
# returns 7
fit <- ppm(swedishpines ~ 1, Strauss(r=7))
reach(fit)
# returns 7

reach(OrdThresh(42))
# returns Inf

reach(MultiStrauss(matrix(c(1,3,3,1),2,2)))
# returns 3
```

reach.dppm *Range of Interaction for a Determinantal Point Process Model*

Description

Returns the range of interaction for a determinantal point process model.

Usage

```
  ## S3 method for class 'dppm'
reach(x, ...)

  ## S3 method for class 'detpointprocfamily'
reach(x, ...)
```

Arguments

x	Model of class "detpointprocfamily" or "dppm".
...	Additional arguments passed to the range function of the given model.

Details

The range of interaction for a determinantal point process model may defined as the smallest number R such that $g(r) = 1$ for all $r \geq R$, where g is the pair correlation function. For many models the range is infinite, but one may instead use a value where the pair correlation function is sufficiently close to 1. For example in the Matern model this defaults to finding R such that $g(R) = 0.99$.

Value

Numeric

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
reach(dppMatern(lambda=100, alpha=.01, nu=1, d=2))
```

`reduced.sample` *Reduced Sample Estimator using Histogram Data*

Description

Compute the Reduced Sample estimator of a survival time distribution function, from histogram data

Usage

```
reduced.sample(nco, cen, ncc, show=FALSE, uppercen=0)
```

Arguments

`nco`	vector of counts giving the histogram of uncensored observations (those survival times that are less than or equal to the censoring time)
`cen`	vector of counts giving the histogram of censoring times
`ncc`	vector of counts giving the histogram of censoring times for the uncensored observations only
`uppercen`	number of censoring times greater than the rightmost histogram breakpoint (if there are any)
`show`	Logical value controlling the amount of detail returned by the function value (see below)

Details

This function is needed mainly for internal use in **spatstat**, but may be useful in other applications where you want to form the reduced sample estimator from a huge dataset.

Suppose T_i are the survival times of individuals $i = 1, \ldots, M$ with unknown distribution function $F(t)$ which we wish to estimate. Suppose these times are right-censored by random censoring times C_i. Thus the observations consist of right-censored survival times $\tilde{T}_i = \min(T_i, C_i)$ and non-censoring indicators $D_i = 1\{T_i \leq C_i\}$ for each i.

If the number of observations M is large, it is efficient to use histograms. Form the histogram `cen` of all censoring times C_i. That is, `obs[k]` counts the number of values C_i in the interval `(breaks[k],breaks[k+1]]` for $k > 1$ and `[breaks[1],breaks[2]]` for $k = 1$. Also form the histogram `nco` of all uncensored times, i.e. those $\tilde{T}_i$ such that $D_i = 1$, and the histogram of all censoring times for which the survival time is uncensored, i.e. those C_i such that $D_i = 1$. These three histograms are the arguments passed to `kaplan.meier`.

The return value `rs` is the reduced-sample estimator of the distribution function $F(t)$. Specifically, `rs[k]` is the reduced sample estimate of `F(breaks[k+1])`. The value is exact, i.e. the use of histograms does not introduce any approximation error.

Note that, for the results to be valid, either the histogram breaks must span the censoring times, or the number of censoring times that do not fall in a histogram cell must have been counted in `uppercen`.

Value

If show = FALSE, a numeric vector giving the values of the reduced sample estimator. If show=TRUE, a list with three components which are vectors of equal length,

rs	Reduced sample estimate of the survival time c.d.f. $F(t)$
numerator	numerator of the reduced sample estimator
denominator	denominator of the reduced sample estimator

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

kaplan.meier, km.rs

---

reflect *Reflect In Origin*

Description

Reflects a geometrical object through the origin.

Usage

```
  reflect(X)

  ## S3 method for class 'im'
reflect(X)

  ## Default S3 method:
reflect(X)
```

Arguments

X	Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), or a window (object of class "owin").

Details

The object X is reflected through the origin. That is, each point in X with coordinates (x, y) is mapped to the position $(-x, -y)$.

This is equivalent to applying the affine transformation with matrix diag(c(-1,-1)). It is also equivalent to rotation about the origin by 180 degrees.

The command reflect is generic, with a method for pixel images and a default method.

Value

Another object of the same type, representing the result of reflection.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

affine, flipxy

Examples

```
plot(reflect(as.im(letterR)))
plot(reflect(letterR), add=TRUE)
```

---

regularpolygon *Create A Regular Polygon*

Description

Create a window object representing a regular (equal-sided) polygon.

Usage

```
regularpolygon(n, edge = 1, centre = c(0, 0), ...,
               align = c("bottom", "top", "left", "right", "no"))

hexagon(edge = 1, centre = c(0,0), ...,
        align = c("bottom", "top", "left", "right", "no"))
```

Arguments

n	Number of edges in the polygon.
edge	Length of each edge in the polygon. A single positive number.
centre	Coordinates of the centre of the polygon. A numeric vector of length 2, or a list(x,y) giving the coordinates of exactly one point, or a point pattern (object of class "ppp") containing exactly one point.
align	Character string specifying whether to align one of the edges with a vertical or horizontal boundary.
...	Ignored.

Details

The function regularpolygon creates a regular (equal-sided) polygon with n sides, centred at centre, with sides of equal length edge. The function hexagon is the special case n=6.

The orientation of the polygon is determined by the argument align. If align="no", one vertex of the polygon is placed on the x-axis. Otherwise, an edge of the polygon is aligned with one side of the frame, specified by the value of align.

Value

A window (object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

disc, ellipse, owin.

hextess for hexagonal tessellations.

Examples

```
plot(hexagon())
plot(regularpolygon(7))
plot(regularpolygon(7, align="left"))
```

---

relevel.im *Reorder Levels of a Factor-Valued Image or Pattern*

Description

For a pixel image with factor values, or a point pattern with factor-valued marks, the levels of the factor are re-ordered so that the level ref is first and the others are moved down.

Usage

```
  ## S3 method for class 'im'
relevel(x, ref, ...)

  ## S3 method for class 'ppp'
relevel(x, ref, ...)

  ## S3 method for class 'ppx'
relevel(x, ref, ...)
```

Arguments

x	A pixel image (object of class "im") with factor values, or a point pattern (object of class "ppp", "ppx", "lpp" or "pp3") with factor-valued marks.
ref	The reference level.
...	Ignored.

Details

These functions are methods for the generic relevel.

If x is a pixel image (object of class "im") with factor values, or a point pattern (object of class "ppp", "ppx", "lpp" or "pp3") with factor-valued marks, the levels of the factor are changed so that the level specified by ref comes first.

Value

Object of the same kind as x.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

mergeLevels

Examples

```
amacrine
relevel(amacrine, "on")
```

---

reload.or.compute *Compute Unless Previously Saved*

Description

If the designated file does not yet exist, evaluate the expression and save the results in the file. If the file already exists, re-load the results from the file.

Usage

```
reload.or.compute(filename, expr, objects = NULL,
                  destination = parent.frame(), force=FALSE)
```

Arguments

filename	Name of data file. A character string.
expr	R language expression to be evaluated.
objects	Optional character vector of names of objects to be saved in filename after evaluating expr, or names of objects that should be present in filename when loaded.
destination	Environment in which the resulting objects should be assigned.
force	Logical value indicating whether to perform the computation in any case.

Details

This facility is useful for saving, and later re-loading, the results of time-consuming computations. It would typically be used in an R script file or an Sweave document.

If the file called filename does not yet exist, then expr will be evaluated and the results will be saved in filename. The optional argument objects specifies which results should be saved to the file: the default is to save all objects that were created by evaluating the expression.

If the file called filename already exists, then it will be loaded. The optional argument objects specifies the names of objects that should be present in the file; a warning is issued if any of them are missing.

The resulting objects can be assigned into any desired destination. The default behaviour is equivalent to evaluating expr in the current environment.

If `force=TRUE` then `expr` will be evaluated (regardless of whether the file already exists or not) and the results will be saved in `filename`, overwriting any previously-existing file with that name. This is a convenient way to force the code to re-compute everything in an R script file or `Sweave` document.

Value

Character vector (invisible) giving the names of the objects computed or loaded.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
## Not run:
 if(FALSE) {
  reload.or.compute("mydata.rda", {
     x <- very.long.computation()
     y <- 42
   })
}

## End(Not run)
```

`relrisk` *Estimate of Spatially-Varying Relative Risk*

Description

Generic command to estimate the spatially-varying probability of each type of point, or the ratios of such probabilities.

Usage

```
relrisk(X, ...)
```

Arguments

`X`	Either a point pattern (class "ppp") or a fitted point process model (class "ppm") from which the probabilities will be estimated.
`...`	Additional arguments appropriate to the method.

Details

In a point pattern containing several different types of points, we may be interested in the spatially-varying probability of each possible type, or the relative risks which are the ratios of such probabilities.

The command `relrisk` is generic and can be used to estimate relative risk in different ways.

The function `relrisk.ppp` is the method for point pattern datasets. It computes *nonparametric* estimates of relative risk by kernel smoothing.

The function relrisk.ppm is the method for fitted point process models (class "ppm"). It computes *parametric* estimates of relative risk, using the fitted model.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

relrisk.ppp, relrisk.ppm.

---

relrisk.ppm *Parametric Estimate of Spatially-Varying Relative Risk*

Description

Given a point process model fitted to a multitype point pattern, this function computes the fitted spatially-varying probability of each type of point, or the ratios of such probabilities, according to the fitted model. Optionally the standard errors of the estimates are also computed.

Usage

```
## S3 method for class 'ppm'
relrisk(X, ...,
                     at = c("pixels", "points"),
                     relative = FALSE, se = FALSE,
                     casecontrol = TRUE, control = 1, case,
                     ngrid = NULL, window = NULL)
```

Arguments

X	A fitted point process model (object of class "ppm").
...	Ignored.
at	String specifying whether to compute the probability values at a grid of pixel locations (at="pixels") or only at the points of X (at="points").
relative	Logical. If FALSE (the default) the algorithm computes the probabilities of each type of point. If TRUE, it computes the *relative risk*, the ratio of probabilities of each type relative to the probability of a control.
se	Logical value indicating whether to compute standard errors as well.
casecontrol	Logical. Whether to treat a bivariate point pattern as consisting of cases and controls, and return only the probability or relative risk of a case. Ignored if there are more than 2 types of points. See Details.
control	Integer, or character string, identifying which mark value corresponds to a control.
case	Integer, or character string, identifying which mark value corresponds to a case (rather than a control) in a bivariate point pattern. This is an alternative to the argument control in a bivariate point pattern. Ignored if there are more than 2 types of points.

ngrid Optional. Dimensions of a rectangular grid of locations inside window where the predictions should be computed. An integer, or an integer vector of length 2, specifying the number of grid points in the y and x directions. (Applies only when at="pixels".)

window Optional. A window (object of class "owin") *delimiting* the locations where predictions should be computed. Defaults to the window of the original data used to fit the model object. (Applies only when at="pixels".)

Details

The command relrisk is generic and can be used to estimate relative risk in different ways.

This function relrisk.ppm is the method for fitted point process models (class "ppm"). It computes *parametric* estimates of relative risk, using the fitted model.

If X is a bivariate point pattern (a multitype point pattern consisting of two types of points) then by default, the points of the first type (the first level of marks(X)) are treated as controls or non-events, and points of the second type are treated as cases or events. Then by default this command computes the spatially-varying *probability* of a case, i.e. the probability $p(u)$ that a point at spatial location u will be a case. If relative=TRUE, it computes the spatially-varying *relative risk* of a case relative to a control, $r(u) = p(u)/(1 - p(u))$.

If X is a multitype point pattern with $m > 2$ types, or if X is a bivariate point pattern and casecontrol=FALSE, then by default this command computes, for each type j, a nonparametric estimate of the spatially-varying *probability* of an event of type j. This is the probability $p_j(u)$ that a point at spatial location u will belong to type j. If relative=TRUE, the command computes the *relative risk* of an event of type j relative to a control, $r_j(u) = p_j(u)/p_k(u)$, where events of type k are treated as controls. The argument control determines which type k is treated as a control.

If at = "pixels" the calculation is performed for every spatial location u on a fine pixel grid, and the result is a pixel image representing the function $p(u)$ or a list of pixel images representing the functions $p_j(u)$ or $r_j(u)$ for $j = 1, \ldots, m$. An infinite value of relative risk (arising because the probability of a control is zero) will be returned as NA.

If at = "points" the calculation is performed only at the data points x_i. By default the result is a vector of values $p(x_i)$ giving the estimated probability of a case at each data point, or a matrix of values $p_j(x_i)$ giving the estimated probability of each possible type j at each data point. If relative=TRUE then the relative risks $r(x_i)$ or $r_j(x_i)$ are returned. An infinite value of relative risk (arising because the probability of a control is zero) will be returned as Inf.

Probabilities and risks are computed from the fitted intensity of the model, using predict.ppm. If se=TRUE then standard errors will also be computed, based on asymptotic theory, using vcov.ppm.

Value

If se=FALSE (the default), the format is described below. If se=TRUE, the result is a list of two entries, estimate and SE, each having the format described below.

If X consists of only two types of points, and if casecontrol=TRUE, the result is a pixel image (if at="pixels") or a vector (if at="points"). The pixel values or vector values are the probabilities of a case if relative=FALSE, or the relative risk of a case (probability of a case divided by the probability of a control) if relative=TRUE.

If X consists of more than two types of points, or if casecontrol=FALSE, the result is:

- (if at="pixels") a list of pixel images, with one image for each possible type of point. The result also belongs to the class "solist" so that it can be printed and plotted.

- (if at="points") a matrix of probabilities, with rows corresponding to data points x_i, and columns corresponding to types j.

The pixel values or matrix entries are the probabilities of each type of point if relative=FALSE, or the relative risk of each type (probability of each type divided by the probability of a control) if relative=TRUE.

If relative=FALSE, the resulting values always lie between 0 and 1. If relative=TRUE, the results are either non-negative numbers, or the values Inf or NA.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

There is another method relrisk.ppp for point pattern datasets which computes *nonparametric* estimates of relative risk by kernel smoothing.

See also relrisk, relrisk.ppp, ppm

Examples

```
fit <- ppm(chorley ~ marks * (x+y))
rr <- relrisk(fit, relative=TRUE, control="lung", se=TRUE)
plot(rr$estimate)
plot(rr$SE)
rrX <- relrisk(fit, at="points", relative=TRUE, control="lung")
```

---

relrisk.ppp *Nonparametric Estimate of Spatially-Varying Relative Risk*

Description

Given a multitype point pattern, this function estimates the spatially-varying probability of each type of point, or the ratios of such probabilities, using kernel smoothing. The default smoothing bandwidth is selected by cross-validation.

Usage

```
## S3 method for class 'ppp'
relrisk(X, sigma = NULL, ..., varcov = NULL, at = "pixels",
          relative=FALSE,
          se=FALSE,
          casecontrol=TRUE, control=1, case)
```

Arguments

X	A multitype point pattern (object of class "ppp" which has factor valued marks).
sigma	Optional. The numeric value of the smoothing bandwidth (the standard deviation of isotropic Gaussian smoothing kernel). Alternatively sigma may be a function which can be used to select a different bandwidth for each type of point. See Details.
...	Arguments passed to bw.relrisk to select the bandwidth, or passed to density.ppp to control the pixel resolution.
varcov	Optional. Variance-covariance matrix of anisotopic Gaussian smoothing kernel. Incompatible with sigma.
at	String specifying whether to compute the probability values at a grid of pixel locations (at="pixels") or only at the points of X (at="points").
relative	Logical. If FALSE (the default) the algorithm computes the probabilities of each type of point. If TRUE, it computes the *relative risk*, the ratio of probabilities of each type relative to the probability of a control.
se	Logical value indicating whether to compute standard errors as well.
casecontrol	Logical. Whether to treat a bivariate point pattern as consisting of cases and controls, and return only the probability or relative risk of a case. Ignored if there are more than 2 types of points. See Details.
control	Integer, or character string, identifying which mark value corresponds to a control.
case	Integer, or character string, identifying which mark value corresponds to a case (rather than a control) in a bivariate point pattern. This is an alternative to the argument control in a bivariate point pattern. Ignored if there are more than 2 types of points.

Details

The command relrisk is generic and can be used to estimate relative risk in different ways.

This function relrisk.ppp is the method for point pattern datasets. It computes *nonparametric* estimates of relative risk by kernel smoothing.

If X is a bivariate point pattern (a multitype point pattern consisting of two types of points) then by default, the points of the first type (the first level of marks(X)) are treated as controls or non-events, and points of the second type are treated as cases or events. Then by default this command computes the spatially-varying *probability* of a case, i.e. the probability $p(u)$ that a point at spatial location u will be a case. If relative=TRUE, it computes the spatially-varying *relative risk* of a case relative to a control, $r(u) = p(u)/(1 - p(u))$.

If X is a multitype point pattern with $m > 2$ types, or if X is a bivariate point pattern and casecontrol=FALSE, then by default this command computes, for each type j, a nonparametric estimate of the spatially-varying *probability* of an event of type j. This is the probability $p_j(u)$ that a point at spatial location u will belong to type j. If relative=TRUE, the command computes the *relative risk* of an event of type j relative to a control, $r_j(u) = p_j(u)/p_k(u)$, where events of type k are treated as controls. The argument control determines which type k is treated as a control.

If at = "pixels" the calculation is performed for every spatial location u on a fine pixel grid, and the result is a pixel image representing the function $p(u)$ or a list of pixel images representing the functions $p_j(u)$ or $r_j(u)$ for $j = 1, \ldots, m$. An infinite value of relative risk (arising because the probability of a control is zero) will be returned as NA.

If `at = "points"` the calculation is performed only at the data points x_i. By default the result is a vector of values $p(x_i)$ giving the estimated probability of a case at each data point, or a matrix of values $p_j(x_i)$ giving the estimated probability of each possible type j at each data point. If `relative=TRUE` then the relative risks $r(x_i)$ or $r_j(x_i)$ are returned. An infinite value of relative risk (arising because the probability of a control is zero) will be returned as `Inf`.

Estimation is performed by a simple Nadaraja-Watson type kernel smoother (Diggle, 2003). The smoothing bandwidth can be specified in any of the following ways:

- `sigma` is a single numeric value, giving the standard deviation of the isotropic Gaussian kernel.
- `sigma` is a numeric vector of length 2, giving the standard deviations in the x and y directions of a Gaussian kernel.
- `varcov` is a 2 by 2 matrix giving the variance-covariance matrix of the Gaussian kernel.
- `sigma` is a `function` which selects the bandwidth. Bandwidth selection will be applied **separately to each type of point**. An example of such a function is `bw.diggle`.
- `sigma` and `varcov` are both missing or null. Then a **common** smoothing bandwidth `sigma` will be selected by cross-validation using `bw.relrisk`.

If `se=TRUE` then standard errors will also be computed, based on asymptotic theory, *assuming a Poisson process*.

Value

If `se=FALSE` (the default), the format is described below. If `se=TRUE`, the result is a list of two entries, `estimate` and `SE`, each having the format described below.

If `X` consists of only two types of points, and if `casecontrol=TRUE`, the result is a pixel image (if `at="pixels"`) or a vector (if `at="points"`). The pixel values or vector values are the probabilities of a case if `relative=FALSE`, or the relative risk of a case (probability of a case divided by the probability of a control) if `relative=TRUE`.

If `X` consists of more than two types of points, or if `casecontrol=FALSE`, the result is:

- (if `at="pixels"`) a list of pixel images, with one image for each possible type of point. The result also belongs to the class `"solist"` so that it can be printed and plotted.
- (if `at="points"`) a matrix of probabilities, with rows corresponding to data points x_i, and columns corresponding to types j.

The pixel values or matrix entries are the probabilities of each type of point if `relative=FALSE`, or the relative risk of each type (probability of each type divided by the probability of a control) if `relative=TRUE`.

If `relative=FALSE`, the resulting values always lie between 0 and 1. If `relative=TRUE`, the results are either non-negative numbers, or the values `Inf` or `NA`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

See Also

There is another method relrisk.ppm for point process models which computes *parametric* estimates of relative risk, using the fitted model.

See also bw.relrisk, density.ppp, Smooth.ppp, eval.im

Examples

```
p.oak <- relrisk(urkiola, 20)
if(interactive()) {
   plot(p.oak, main="proportion of oak")
   plot(eval.im(p.oak > 0.3), main="More than 30 percent oak")
   plot(split(lansing), main="Lansing Woods")
   p.lan <- relrisk(lansing, 0.05, se=TRUE)
   plot(p.lan$estimate, main="Lansing Woods species probability")
   plot(p.lan$SE, main="Lansing Woods standard error")
   wh <- im.apply(p.lan$estimate, which.max)
   types <- levels(marks(lansing))
   wh <- eval.im(types[wh])
   plot(wh, main="Most common species")
}
```

Replace.im *Reset Values in Subset of Image*

Description

Reset the values in a subset of a pixel image.

Usage

```
  ## S3 replacement method for class 'im'
x[i, j] <- value
```

Arguments

x	A two-dimensional pixel image. An object of class "im".
i	Object defining the subregion or subset to be replaced. Either a spatial window (an object of class "owin"), or a pixel image with logical values, or a point pattern (an object of class "ppp"), or any type of index that applies to a matrix, or something that can be converted to a point pattern by as.ppp (using the window of x).
j	An integer or logical vector serving as the column index if matrix indexing is being used. Ignored if i is appropriate to some sort of replacement *other than* matrix indexing.
value	Vector, matrix, factor or pixel image containing the replacement values. Short vectors will be recycled.

Details

This function changes some of the pixel values in a pixel image. The image x must be an object of class "im" representing a pixel image defined inside a rectangle in two-dimensional space (see im.object).

The subset to be changed is determined by the arguments i,j according to the following rules (which are checked in this order):

1. i is a spatial object such as a window, a pixel image with logical values, or a point pattern; or
2. i,j are indices for the matrix as.matrix(x); or
3. i can be converted to a point pattern by as.ppp(i, W=Window(x)), and i is not a matrix.

If i is a spatial window (an object of class "owin"), the values of the image inside this window are changed.

If i is a point pattern (an object of class "ppp"), then the values of the pixel image at the points of this pattern are changed.

If i does not satisfy any of the conditions above, then the algorithm tries to interpret i,j as indices for the matrix as.matrix(x). Either i or j may be missing or blank.

If none of the conditions above are met, and if i is not a matrix, then i is converted into a point pattern by as.ppp(i, W=Window(x)). Again the values of the pixel image at the points of this pattern are changed.

Value

The image x with the values replaced.

Warning

If you have a 2-column matrix containing the x, y coordinates of point locations, then to prevent this being interpreted as an array index, you should convert it to a data.frame or to a point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

im.object, [.im, [, ppp.object, as.ppp, owin.object

Examples

```
# make up an image
X <- setcov(unit.square())
plot(X)

# a rectangular subset
W <- owin(c(0,0.5),c(0.2,0.8))
X[W] <- 2
plot(X)

# a polygonal subset
data(letterR)
```

```
R <- affine(letterR, diag(c(1,1)/2), c(-2,-0.7))
X[R] <- 3
plot(X)

# a point pattern
P <- rpoispp(20)
X[P] <- 10
plot(X)

# change pixel value at a specific location
X[list(x=0.1,y=0.2)] <- 7

# matrix indexing --- single vector index
X[1:2570] <- 10
plot(X)

# matrix indexing using double indices
X[1:257,1:10] <- 5
plot(X)

# matrix indexing using a matrix of indices
X[cbind(1:257,1:257)] <- 10
X[cbind(257:1,1:257)] <- 10
plot(X)
```

Replace.linim *Reset Values in Subset of Image on Linear Network*

Description

Reset the values in a subset of a pixel image on a linear network.

Usage

```
## S3 replacement method for class 'linim'
x[i, j] <- value
```

Arguments

x	A pixel image on a linear network. An object of class "linim".
i	Object defining the subregion or subset to be replaced. Either a spatial window (an object of class "owin"), or a pixel image with logical values, or a point pattern (an object of class "ppp"), or any type of index that applies to a matrix, or something that can be converted to a point pattern by as.ppp (using the window of x).
j	An integer or logical vector serving as the column index if matrix indexing is being used. Ignored if i is appropriate to some sort of replacement *other than* matrix indexing.
value	Vector, matrix, factor or pixel image containing the replacement values. Short vectors will be recycled.

Details

This function changes some of the pixel values in a pixel image. The image x must be an object of class "linim" representing a pixel image on a linear network.

The pixel values are replaced according to the rules described in the help for [<-.im. Then the auxiliary data are updated.

Value

The image x with the values replaced.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

[<-.im.

Examples

```
# make a function
Y <- as.linim(distfun(runiflpp(5, simplenet)))
# replace some values
B <- square(c(0.25, 0.55))
Y[B] <- 2
plot(Y, main="")
plot(B, add=TRUE, lty=3)
X <- runiflpp(4, simplenet)
Y[X] <- 5
```

---

requireversion *Require a Specific Version of a Package*

Description

Checks that the version number of a specified package is greater than or equal to the specified version number. For use in stand-alone R scripts.

Usage

```
requireversion(pkg, ver)
```

Arguments

pkg	Package name.
ver	Character string containing version number.

Details

This function checks whether the installed version of the package pkg is greater than or equal to ver.

It is useful in stand-alone R scripts, which often require a particular version of a package in order to work correctly.

This function should not be used inside a package: for that purpose, the dependence on packages and versions should be specified in the package description file.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Examples

```
  ## Not run:
    requireversion(spatstat, "1.42-0")

## End(Not run)
```

---

rescale *Convert dataset to another unit of length*

Description

Converts between different units of length in a spatial dataset, such as a point pattern or a window.

Usage

```
rescale(X, s, unitname)
```

Arguments

X	Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), or a window (object of class "owin").
s	Conversion factor: the new units are s times the old units.
unitname	Optional. New name for the unit of length. See unitname.

Details

This is generic. Methods are provided for many spatial objects.

The spatial coordinates in the dataset X will be re-expressed in terms of a new unit of length that is s times the current unit of length given in X. The name of the unit of length will also be adjusted. The result is an object of the same type, representing the same data, but expressed in the new units.

For example if X is a dataset giving coordinates in metres, then rescale(X,1000) will take the new unit of length to be 1000 metres. To do this, it will divide the old coordinate values by 1000 to obtain coordinates expressed in kilometres, and change the name of the unit of length from "metres" to "1000 metres".

If unitname is given, it will be taken as the new name of the unit of length. It should be a valid name for the unit of length, as described in the help for unitname. For example if X is a dataset giving coordinates in metres, rescale(X, 1000, "km") will divide the coordinate values by 1000 to obtain coordinates in kilometres, and the unit name will be changed to "km".

Value

Another object of the same type, representing the same data, but expressed in the new units.

Note

The result of this operation is equivalent to the original dataset. If you want to actually change the coordinates by a linear transformation, producing a dataset that is not equivalent to the original one, use affine.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

Available methods: rescale.im, rescale.layered, rescale.linnet, rescale.lpp, rescale.owin, rescale.ppp, rescale.psp and rescale.unitname.

Other generics: unitname, affine, rotate, shift.

---

rescale.im *Convert Pixel Image to Another Unit of Length*

Description

Converts a pixel image to another unit of length.

Usage

```
## S3 method for class 'im'
rescale(X, s, unitname)
```

Arguments

X	Pixel image (object of class "im").
s	Conversion factor: the new units are s times the old units.
unitname	Optional. New name for the unit of length. See unitname.

Details

This is a method for the generic function `rescale`.

The spatial coordinates of the pixels in `X` will be re-expressed in terms of a new unit of length that is `s` times the current unit of length given in `X`. (Thus, the coordinate values are *divided* by `s`, while the unit value is multiplied by `s`).

If `s` is missing, then the coordinates will be re-expressed in 'native' units; for example if the current unit is equal to 0.1 metres, then the coordinates will be re-expressed in metres.

The result is a pixel image representing the *same* data but re-expressed in a different unit.

Pixel values are unchanged. This may not be what you intended!

Value

Another pixel image (of class `"im"`), containing the same pixel values, but with pixel coordinates expressed in the new units.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`im`, `rescale`, `unitname`, `eval.im`

Examples

```
# Bramble Canes data: 1 unit = 9 metres
  data(bramblecanes)
# distance transform
  Z <- distmap(bramblecanes)
# convert to metres
# first alter the pixel values
  Zm <- eval.im(9 * Z)
# now rescale the pixel coordinates
  Z <- rescale(Zm, 1/9)
# or equivalently
  Z <- rescale(Zm)
```

`rescale.owin` *Convert Window to Another Unit of Length*

Description

Converts a window to another unit of length.

Usage

```
  ## S3 method for class 'owin'
rescale(X, s, unitname)
```

Arguments

X	Window (object of class "owin").
s	Conversion factor: the new units are s times the old units.
unitname	Optional. New name for the unit of length. See unitname.

Details

This is a method for the generic function rescale.

The spatial coordinates in the window X (and its window) will be re-expressed in terms of a new unit of length that is s times the current unit of length given in X. (Thus, the coordinate values are *divided* by s, while the unit value is multiplied by s).

The result is a window representing the *same* region of space, but re-expressed in a different unit.

If s is missing, then the coordinates will be re-expressed in 'native' units; for example if the current unit is equal to 0.1 metres, then the coordinates will be re-expressed in metres.

Value

Another window object (of class "owin") representing the same window, but expressed in the new units.

Note

The result of this operation is equivalent to the original window. If you want to actually change the coordinates by a linear transformation, producing a window that is larger or smaller than the original one, use affine.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

unitname, rescale, rescale.owin, affine, rotate, shift

Examples

```
  data(swedishpines)
  W <- Window(swedishpines)
  W
# coordinates are in decimetres (0.1 metre)
# convert to metres:
  rescale(W, 10)
# or equivalently
  rescale(W)
```

rescale.ppp *Convert Point Pattern to Another Unit of Length*

Description

Converts a point pattern dataset to another unit of length.

Usage

```
## S3 method for class 'ppp'
rescale(X, s, unitname)
```

Arguments

X	Point pattern (object of class "ppp").
s	Conversion factor: the new units are s times the old units.
unitname	Optional. New name for the unit of length. See unitname.

Details

This is a method for the generic function rescale.

The spatial coordinates in the point pattern X (and its window) will be re-expressed in terms of a new unit of length that is s times the current unit of length given in X. (Thus, the coordinate values are *divided* by s, while the unit value is multiplied by s).

The result is a point pattern representing the *same* data but re-expressed in a different unit.

Mark values are unchanged.

If s is missing, then the coordinates will be re-expressed in 'native' units; for example if the current unit is equal to 0.1 metres, then the coordinates will be re-expressed in metres.

Value

Another point pattern (of class "ppp"), representing the same data, but expressed in the new units.

Note

The result of this operation is equivalent to the original point pattern. If you want to actually change the coordinates by a linear transformation, producing a point pattern that is not equivalent to the original one, use affine.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

unitname, rescale, rescale.owin, affine, rotate, shift

Examples

```
# Bramble Canes data: 1 unit = 9 metres
  data(bramblecanes)
# convert to metres
  bram <- rescale(bramblecanes, 1/9)
# or equivalently
  bram <- rescale(bramblecanes)
```

---

rescale.psp *Convert Line Segment Pattern to Another Unit of Length*

Description

Converts a line segment pattern dataset to another unit of length.

Usage

```
 ## S3 method for class 'psp'
rescale(X, s, unitname)
```

Arguments

X	Line segment pattern (object of class "psp").
s	Conversion factor: the new units are s times the old units.
unitname	Optional. New name for the unit of length. See unitname.

Details

This is a method for the generic function rescale.

The spatial coordinates in the line segment pattern X (and its window) will be re-expressed in terms of a new unit of length that is s times the current unit of length given in X. (Thus, the coordinate values are *divided* by s, while the unit value is multiplied by s).

The result is a line segment pattern representing the *same* data but re-expressed in a different unit.

Mark values are unchanged.

If s is missing, then the coordinates will be re-expressed in 'native' units; for example if the current unit is equal to 0.1 metres, then the coordinates will be re-expressed in metres.

Value

Another line segment pattern (of class "psp"), representing the same data, but expressed in the new units.

Note

The result of this operation is equivalent to the original segment pattern. If you want to actually change the coordinates by a linear transformation, producing a segment pattern that is not equivalent to the original one, use affine.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

units, affine, rotate, shift

Examples

```
data(copper)
X <- copper$Lines
X
# data are in km
# convert to metres
rescale(X, 1/1000)

# convert data and rename unit
rescale(X, 1/1000, c("metre", "metres"))
```

`rescue.rectangle` *Convert Window Back To Rectangle*

Description

Determines whether the given window is really a rectangle aligned with the coordinate axes, and if so, converts it to a rectangle object.

Usage

```
rescue.rectangle(W)
```

Arguments

W	A window (object of class "owin").

Details

This function decides whether the window W is actually a rectangle aligned with the coordinate axes. This will be true if W is

- a rectangle (window object of type "rectangle");
- a polygon (window object of type "polygonal" with a single polygonal boundary) that is a rectangle aligned with the coordinate axes;
- a binary mask (window object of type "mask") in which all the pixel entries are TRUE.

If so, the function returns this rectangle, a window object of type "rectangle". If not, the function returns W.

Value

Another object of class "owin" representing the same window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

as.owin, owin.object

Examples

```
w <- owin(poly=list(x=c(0,1,1,0),y=c(0,0,1,1)))
rw <- rescue.rectangle(w)

w <- as.mask(unit.square())
rw <- rescue.rectangle(w)
```

---

residuals.dppm *Residuals for Fitted Determinantal Point Process Model*

Description

Given a determinantal point process model fitted to a point pattern, compute residuals.

Usage

```
## S3 method for class 'dppm'
residuals(object, ...)
```

Arguments

object	The fitted determinatal point process model (an object of class "dppm") for which residuals should be calculated.
...	Arguments passed to residuals.ppm.

Details

This function extracts the intensity component of the model using as.ppm and then applies residuals.ppm to compute the residuals.

Use plot.msr to plot the residuals directly.

Value

An object of class "msr" representing a signed measure or vector-valued measure (see msr). This object can be plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

msr, dppm

Examples

```
fit <- dppm(swedishpines ~ x, dppGauss())
rr <- residuals(fit)
```

---

residuals.kppm *Residuals for Fitted Cox or Cluster Point Process Model*

Description

Given a Cox or cluster point process model fitted to a point pattern, compute residuals.

Usage

```
  ## S3 method for class 'kppm'
residuals(object, ...)
```

Arguments

object	The fitted point process model (an object of class "kppm") for which residuals should be calculated.
...	Arguments passed to residuals.ppm.

Details

This function extracts the intensity component of the model using as.ppm and then applies residuals.ppm to compute the residuals.

Use plot.msr to plot the residuals directly.

Value

An object of class "msr" representing a signed measure or vector-valued measure (see msr). This object can be plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

msr, kppm

Examples

```
fit <- kppm(redwood ~ x, "Thomas")
rr <- residuals(fit)
```

residuals.mppm *Residuals for Point Process Model Fitted to Multiple Point Patterns*

Description

Given a point process model fitted to multiple point patterns, compute residuals for each pattern.

Usage

```
## S3 method for class 'mppm'
residuals(object, type = "raw", ...,
                              fittedvalues = fitted.mppm(object))
```

Arguments

object	Fitted point process model (object of class "mppm").
...	Ignored.
type	Type of residuals: either "raw", "pearson" or "inverse". Partially matched.
fittedvalues	Advanced use only. Fitted values of the model to be used in the calculation.

Details

Baddeley et al (2005) defined residuals for the fit of a point process model to spatial point pattern data. For an explanation of these residuals, see the help file for residuals.ppm.

This function computes the residuals for a point process model fitted to *multiple* point patterns. The object should be an object of class "mppm" obtained from mppm.

The return value is a list. The number of entries in the list equals the number of point patterns in the original data. Each entry in the list has the same format as the output of residuals.ppm. That is, each entry in the list is a signed measure (object of class "msr") giving the residual measure for the corresponding point pattern.

Value

A list of signed measures (objects of class "msr") giving the residual measure for each of the original point patterns. See Details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A., Turner, R., Moller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

See Also

mppm, residuals.mppm

Examples

```
fit <- mppm(Bugs ~ x, hyperframe(Bugs=waterstriders))
r <- residuals(fit)
# compute total residual for each point pattern
rtot <- sapply(r, integral.msr)
# standardise the total residuals
areas <- sapply(windows.mppm(fit), area.owin)
rtot/sqrt(areas)
```

---

residuals.ppm — *Residuals for Fitted Point Process Model*

Description

Given a point process model fitted to a point pattern, compute residuals.

Usage

```
  ## S3 method for class 'ppm'
residuals(object, type="raw", ...,
                check=TRUE, drop=FALSE,
                fittedvalues=NULL,
                new.coef=NULL, dropcoef=FALSE,
                quad=NULL)
```

Arguments

object	The fitted point process model (an object of class "ppm") for which residuals should be calculated.
type	String indicating the type of residuals to be calculated. Current options are "raw", "inverse", "pearson" and "score". A partial match is adequate.
...	Ignored.
check	Logical value indicating whether to check the internal format of object. If there is any possibility that this object has been restored from a dump file, or has otherwise lost track of the environment where it was originally computed, set check=TRUE.
drop	Logical value determining whether to delete quadrature points that were not used to fit the model. See quad.ppm for explanation.
fittedvalues	Vector of fitted values for the conditional intensity at the quadrature points, from which the residuals will be computed. For expert use only.
new.coef	Optional. Numeric vector of coefficients for the model, replacing coef(object). See the section on Modified Residuals below.
dropcoef	Internal use only.
quad	Optional. Data specifying how to re-fit the model. A list of arguments passed to quadscheme. See the section on Modified Residuals below.

Details

This function computes several kinds of residuals for the fit of a point process model to a spatial point pattern dataset (Baddeley et al, 2005). Use `plot.msr` to plot the residuals directly, or `diagnose.ppm` to produce diagnostic plots based on these residuals.

The argument `object` must be a fitted point process model (object of class `"ppm"`). Such objects are produced by the maximum pseudolikelihood fitting algorithm `ppm`. This fitted model object contains complete information about the original data pattern.

Residuals are attached both to the data points and to some other points in the window of observation (namely, to the dummy points of the quadrature scheme used to fit the model). If the fitted model is correct, then the sum of the residuals over all (data and dummy) points in a spatial region B has mean zero. For further explanation, see Baddeley et al (2005).

The type of residual is chosen by the argument `type`. Current options are

`"raw"`: the raw residuals

$$r_j = z_j - w_j \lambda_j$$

at the quadrature points u_j, where z_j is the indicator equal to 1 if u_j is a data point and 0 if u_j is a dummy point; w_j is the quadrature weight attached to u_j; and

$$\lambda_j = \hat{\lambda}(u_j, x)$$

is the conditional intensity of the fitted model at u_j. These are the spatial analogue of the martingale residuals of a one-dimensional counting process.

`"inverse"`: the 'inverse-lambda' residuals (Baddeley et al, 2005)

$$r_j^{(I)} = \frac{r_j}{\lambda_j} = \frac{z_j}{\lambda_j} - w_j$$

obtained by dividing the raw residuals by the fitted conditional intensity. These are a counterpart of the exponential energy marks (see `eem`).

`"pearson"`: the Pearson residuals (Baddeley et al, 2005)

$$r_j^{(P)} = \frac{r_j}{\sqrt{\lambda_j}} = \frac{z_j}{\sqrt{\lambda_j}} - w_j \sqrt{\lambda_j}$$

obtained by dividing the raw residuals by the square root of the fitted conditional intensity. The Pearson residuals are standardised, in the sense that if the model (true and fitted) is Poisson, then the sum of the Pearson residuals in a spatial region B has variance equal to the area of B.

`"score"`: the score residuals (Baddeley et al, 2005)

$$r_j = (z_j - w_j \lambda_j) x_j$$

obtained by multiplying the raw residuals r_j by the covariates x_j for quadrature point j. The score residuals always sum to zero.

The result of `residuals.ppm` is a measure (object of class `"msr"`). Use `plot.msr` to plot the residuals directly, or `diagnose.ppm` to produce diagnostic plots based on these residuals. Use `integral.msr` to compute the total residual.

By default, the window of the measure is the same as the original window of the data. If `drop=TRUE` then the window is the domain of integration of the pseudolikelihood or composite likelihood. This only matters when the model `object` was fitted using the border correction: in that case, if `drop=TRUE` the window of the residuals is the erosion of the original data window by the border correction distance `rbord`.

Value

An object of class "msr" representing a signed measure or vector-valued measure (see msr). This object can be plotted.

Modified Residuals

Sometimes we want to modify the calculation of residuals by using different values for the model parameters. This capability is provided by the arguments new.coef and quad.

If new.coef is given, then the residuals will be computed by taking the model parameters to be new.coef. This should be a numeric vector of the same length as the vector of fitted model parameters coef(object).

If new.coef is missing and quad is given, then the model parameters will be determined by refitting the model using a new quadrature scheme specified by quad. Residuals will be computed for the original model object using these new parameter values.

The argument quad should normally be a list of arguments in name=value format that will be passed to quadscheme (together with the original data points) to determine the new quadrature scheme. It may also be a quadrature scheme (object of class "quad") to which the model should be fitted, or a point pattern (object of class "ppp") specifying the *dummy points* in a new quadrature scheme.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

Baddeley, A., Møller, J. and Pakes, A.G. (2008) Properties of residuals for spatial point processes. *Annals of the Institute of Statistical Mathematics* **60**, 627–649.

See Also

msr, diagnose.ppm, ppm.object, ppm

Examples

```
fit <- ppm(cells, ~x, Strauss(r=0.15))

# Pearson residuals
rp <- residuals(fit, type="pe")
rp

# simulated data
X <- rStrauss(100,0.7,0.05)
# fit Strauss model
fit <- ppm(X, ~1, Strauss(0.05))
res.fit <- residuals(fit)

# check that total residual is 0
integral.msr(residuals(fit, drop=TRUE))

# true model parameters
```

```
truecoef <- c(log(100), log(0.7))
res.true <- residuals(fit, new.coef=truecoef)
```

---

rex *Richardson Extrapolation*

Description

Performs Richardson Extrapolation on a sequence of approximate values.

Usage

```
rex(x, r = 2, k = 1, recursive = FALSE)
```

Arguments

x	A numeric vector or matrix, whose columns are successive estimates or approximations to a vector of parameters.
r	A number greater than 1. The ratio of successive step sizes. See Details.
k	Integer. The order of convergence assumed. See Details.
recursive	Logical value indicating whether to perform one step of Richardson extrapolation (`recursive=FALSE`, the default) or repeat the extrapolation procedure until a best estimate is obtained (`recursive=TRUE`.

Details

Richardson extrapolation is a general technique for improving numerical approximations, often used in numerical integration (Brezinski and Zaglia, 1991). It can also be used to improve parameter estimates in statistical models (Baddeley and Turner, 2014).

The successive columns of `x` are assumed to have been obtained using approximations with step sizes $a, a/r, a/r^2, \ldots$ where a is the initial step size (which does not need to be specified).

Estimates based on a step size s are assumed to have an error of order s^k.

Thus, the default values `r=2` and `k=1` imply that the errors in the second column of `x` should be roughly $(1/r)^k = 1/2$ as large as the errors in the first column, and so on.

Value

A matrix whose columns contain a sequence of improved estimates.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`> and Rolf Turner <`r.turner@auckland.ac.nz`>.

References

Baddeley, A. and Turner, R. (2014) Bias correction for parameter estimates of spatial point process models. *Journal of Statistical Computation and Simulation* **84**, 1621–1643. DOI: 10.1080/00949655.2012.755976

Brezinski, C. and Zaglia, M.R. (1991) *Extrapolation Methods. Theory and Practice*. North-Holland.

See Also

bc

Examples

```
# integrals of sin(x) and cos(x) from 0 to pi
# correct answers: 2, 0
est <- function(nsteps) {
  xx <- seq(0, pi, length=nsteps)
  ans <- pi * c(mean(sin(xx)), mean(cos(xx)))
  names(ans) <- c("sin", "cos")
  ans
}
X <- cbind(est(10), est(20), est(40))
X
rex(X)
rex(X, recursive=TRUE)

# fitted Gibbs point process model
fit0 <- ppm(cells ~ 1, Strauss(0.07), nd=16)
fit1 <- update(fit0, nd=32)
fit2 <- update(fit0, nd=64)
co <- cbind(coef(fit0), coef(fit1), coef(fit2))
co
rex(co, k=2, recursive=TRUE)
```

---

rGaussPoisson *Simulate Gauss-Poisson Process*

Description

Generate a random point pattern, a simulated realisation of the Gauss-Poisson Process.

Usage

```
rGaussPoisson(kappa, r, p2, win = owin(c(0,1),c(0,1)),
                ..., nsim=1, drop=TRUE)
```

Arguments

kappa	Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.
r	Diameter of each cluster that consists of exactly 2 points.
p2	Probability that a cluster contains exactly 2 points.
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin.
...	Ignored.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This algorithm generates a realisation of the Gauss-Poisson point process inside the window `win`. The process is constructed by first generating a Poisson point process of parent points with intensity `kappa`. Then each parent point is either retained (with probability `1 - p2`) or replaced by a pair of points at a fixed distance `r` apart (with probability `p2`). In the case of clusters of 2 points, the line joining the two points has uniform random orientation.

In this implementation, parent points are not restricted to lie in the window; the parent process is effectively the uniform Poisson process on the infinite plane.

Value

A point pattern (an object of class `"ppp"`) if `nsim=1`, or a list of point patterns if `nsim > 1`.

Additionally, some intermediate results of the simulation are returned as attributes of the point pattern. See `rNeymanScott`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`rpoispp`, `rThomas`, `rMatClust`, `rNeymanScott`

Examples

```
pp <- rGaussPoisson(30, 0.07, 0.5)
```

`rgbim` *Create Colour-Valued Pixel Image*

Description

Creates an object of class `"im"` representing a two-dimensional pixel image whose pixel values are colours.

Usage

```
rgbim(R, G, B, A, maxColorValue=255, autoscale=FALSE)
hsvim(H, S, V, A, autoscale=FALSE)
```

Arguments

`R,G,B`	Pixel images (objects of class `"im"`) or constants giving the red, green, and blue components of a colour, respectively.
`A`	Optional. Pixel image or constant value giving the alpha (transparency) component of a colour.
`maxColorValue`	Maximum colour channel value for `R,G,B,A`.
`H,S,V`	Pixel images (objects of class `"im"`) or constants giving the hue, saturation, and value components of a colour, respectively.

autoscale — Logical. If TRUE, input values are automatically rescaled to fit the permitted range. RGB values are scaled to lie between 0 and maxColorValue. HSV values are scaled to lie between 0 and 1.

Details

These functions take three pixel images, with real or integer pixel values, and create a single pixel image whose pixel values are colours recognisable to R.

Some of the arguments may be constant numeric values, but at least one of the arguments must be a pixel image. The image arguments should be compatible (in array dimension and in spatial position).

rgbim calls rgb to compute the colours, while hsvim calls hsv. See the help for the relevant function for more information about the meaning of the colour channels.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

im.object, rgb, hsv.

See colourtools for additional colour tools.

Examples

```
# create three images with values in [0,1]
X <- setcov(owin())
X <- eval.im(pmin(1,X))
M <- Window(X)
Y <- as.im(function(x,y){(x+1)/2}, W=M)
Z <- as.im(function(x,y){(y+1)/2}, W=M)
RGB <- rgbim(X, Y, Z, maxColorValue=1)
HSV <- hsvim(X, Y, Z)
plot(RGB, valuesAreColours=TRUE)
plot(HSV, valuesAreColours=TRUE)
```

---

rHardcore — *Perfect Simulation of the Hardcore Process*

Description

Generate a random pattern of points, a simulated realisation of the Hardcore process, using a perfect simulation algorithm.

Usage

```
rHardcore(beta, R = 0, W = owin(), expand=TRUE, nsim=1, drop=TRUE)
```

Arguments

beta	intensity parameter (a positive number).
R	hard core distance (a non-negative number).
W	window (object of class "owin") in which to generate the random pattern. Currently this must be a rectangular window.
expand	Logical. If FALSE, simulation is performed in the window W, which must be rectangular. If TRUE (the default), simulation is performed on a larger window, and the result is clipped to the original window W. Alternatively expand can be an object of class "rmhexpand" (see rmhexpand) determining the expansion method.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates a realisation of the Hardcore point process in the window W using a 'perfect simulation' algorithm.

The Hardcore process is a model for strong spatial inhibition. Two points of the process are forbidden to lie closer than R units apart. The Hardcore process is the special case of the Strauss process (see rStrauss) with interaction parameter γ equal to zero.

The simulation algorithm used to generate the point pattern is 'dominated coupling from the past' as implemented by Berthelsen and Møller (2002, 2003). This is a 'perfect simulation' or 'exact simulation' algorithm, so called because the output of the algorithm is guaranteed to have the correct probability distribution exactly (unlike the Metropolis-Hastings algorithm used in rmh, whose output is only approximately correct).

There is a tiny chance that the algorithm will run out of space before it has terminated. If this occurs, an error message will be generated.

Value

If nsim = 1, a point pattern (object of class "ppp"). If nsim > 1, a list of point patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

based on original code for the Strauss process by Kasper Klitgaard Berthelsen.

References

Berthelsen, K.K. and Møller, J. (2002) A primer on perfect simulation for spatial point processes. *Bulletin of the Brazilian Mathematical Society* 33, 351-367.

Berthelsen, K.K. and Møller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* 30, 549-564.

Møller, J. and Waagepetersen, R. (2003). *Statistical Inference and Simulation for Spatial Point Processes.* Chapman and Hall/CRC.

See Also

rmh, Hardcore, rStrauss, rStraussHard, rDiggleGratton. rDGS, rPenttinen.

Examples

```
X <- rHardcore(0.05,1.5,square(141.4))
Z <- rHardcore(100,0.05)
```

---

rho2hat *Smoothed Relative Density of Pairs of Covariate Values*

Description

Given a point pattern and two spatial covariates Z_1 and Z_2, construct a smooth estimate of the relative risk of the pair (Z_1, Z_2).

Usage

```
rho2hat(object, cov1, cov2, ..., method=c("ratio", "reweight"))
```

Arguments

object	A point pattern (object of class "ppp"), a quadrature scheme (object of class "quad") or a fitted point process model (object of class "ppm").
cov1,cov2	The two covariates. Each argument is either a function(x,y) or a pixel image (object of class "im") providing the values of the covariate at any location, or one of the strings "x" or "y" signifying the Cartesian coordinates.
...	Additional arguments passed to density.ppp to smooth the scatterplots.
method	Character string determining the smoothing method. See Details.

Details

This is a bivariate version of rhohat.

If object is a point pattern, this command produces a smoothed version of the scatterplot of the values of the covariates cov1 and cov2 observed at the points of the point pattern.

The covariates cov1,cov2 must have continuous values.

If object is a fitted point process model, suppose X is the original data point pattern to which the model was fitted. Then this command assumes X is a realisation of a Poisson point process with intensity function of the form

$$\lambda(u) = \rho(Z_1(u), Z_2(u))\kappa(u)$$

where $\kappa(u)$ is the intensity of the fitted model object, and $\rho(z_1, z_2)$ is a function to be estimated. The algorithm computes a smooth estimate of the function ρ.

The method determines how the density estimates will be combined to obtain an estimate of $\rho(z_1, z_2)$:

- If method="ratio", then $\rho(z_1, z_2)$ is estimated by the ratio of two density estimates. The numerator is a (rescaled) density estimate obtained by smoothing the points $(Z_1(y_i), Z_2(y_i))$ obtained by evaluating the two covariate Z_1, Z_2 at the data points y_i. The denominator is a density estimate of the reference distribution of (Z_1, Z_2).
- If method="reweight", then $\rho(z_1, z_2)$ is estimated by applying density estimation to the points $(Z_1(y_i), Z_2(y_i))$ obtained by evaluating the two covariate Z_1, Z_2 at the data points y_i, with weights inversely proportional to the reference density of (Z_1, Z_2).

Value

A pixel image (object of class "im"). Also belongs to the special class "rho2hat" which has a plot method.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Baddeley, A., Chang, Y.-M., Song, Y. and Turner, R. (2012) Nonparametric estimation of the dependence of a point process on spatial covariates. *Statistics and Its Interface* **5** (2), 221–236.

See Also

rhohat, methods.rho2hat

Examples

```
  data(bei)
  attach(bei.extra)
  plot(rho2hat(bei, elev, grad))
  fit <- ppm(bei, ~elev, covariates=bei.extra)
  ## Not run:
  plot(rho2hat(fit, elev, grad))

## End(Not run)
  plot(rho2hat(fit, elev, grad, method="reweight"))
```

---

rhohat *Smoothing Estimate of Intensity as Function of a Covariate*

Description

Computes a smoothing estimate of the intensity of a point process, as a function of a (continuous) spatial covariate.

Usage

```
rhohat(object, covariate, ...)

## S3 method for class 'ppp'
rhohat(object, covariate, ...,
       baseline=NULL, weights=NULL,
       method=c("ratio", "reweight", "transform"),
       horvitz=FALSE,
       smoother=c("kernel", "local"),
       subset=NULL,
       dimyx=NULL, eps=NULL,
       n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
       bwref=bw,
       covname, confidence=0.95)
```

```
## S3 method for class 'quad'
rhohat(object, covariate, ...,
       baseline=NULL, weights=NULL,
       method=c("ratio", "reweight", "transform"),
       horvitz=FALSE,
       smoother=c("kernel", "local"),
       subset=NULL,
       dimyx=NULL, eps=NULL,
       n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
       bwref=bw,
       covname, confidence=0.95)

## S3 method for class 'ppm'
rhohat(object, covariate, ...,
       weights=NULL,
       method=c("ratio", "reweight", "transform"),
       horvitz=FALSE,
       smoother=c("kernel", "local"),
       subset=NULL,
       dimyx=NULL, eps=NULL,
       n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
       bwref=bw,
       covname, confidence=0.95)

## S3 method for class 'lpp'
rhohat(object, covariate, ...,
       weights=NULL,
       method=c("ratio", "reweight", "transform"),
       horvitz=FALSE,
       smoother=c("kernel", "local"),
       subset=NULL,
       nd=1000, eps=NULL, random=TRUE,
       n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
       bwref=bw,
       covname, confidence=0.95)

## S3 method for class 'lppm'
rhohat(object, covariate, ...,
       weights=NULL,
       method=c("ratio", "reweight", "transform"),
       horvitz=FALSE,
       smoother=c("kernel", "local"),
       subset=NULL,
       nd=1000, eps=NULL, random=TRUE,
       n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
       bwref=bw,
       covname, confidence=0.95)
```

Arguments

`object` A point pattern (object of class "ppp" or "lpp"), a quadrature scheme (object of class "quad") or a fitted point process model (object of class "ppm" or "lppm").

covariate	Either a `function(x,y)` or a pixel image (object of class "im") providing the values of the covariate at any location. Alternatively one of the strings "x" or "y" signifying the Cartesian coordinates.
weights	Optional weights attached to the data points. Either a numeric vector of weights for each data point, or a pixel image (object of class "im") or a `function(x,y)` providing the weights.
baseline	Optional baseline for intensity function. A `function(x,y)` or a pixel image (object of class "im") providing the values of the baseline at any location.
method	Character string determining the smoothing method. See Details.
horvitz	Logical value indicating whether to use Horvitz-Thompson weights. See Details.
smoother	Character string determining the smoothing algorithm. See Details.
subset	Optional. A spatial window (object of class "owin") specifying a subset of the data, from which the estimate should be calculated.
dimyx,eps,nd,random	Arguments controlling the pixel resolution at which the covariate will be evaluated. See Details.
bw	Smoothing bandwidth or bandwidth rule (passed to `density.default`).
adjust	Smoothing bandwidth adjustment factor (passed to `density.default`).
n, from, to	Arguments passed to `density.default` to control the number and range of values at which the function will be estimated.
bwref	Optional. An alternative value of `bw` to use when smoothing the reference density (the density of the covariate values observed at all locations in the window).
...	Additional arguments passed to `density.default` or `locfit`.
covname	Optional. Character string to use as the name of the covariate.
confidence	Confidence level for confidence intervals. A number between 0 and 1.

Details

This command estimates the relationship between point process intensity and a given spatial covariate. Such a relationship is sometimes called a *resource selection function* (if the points are organisms and the covariate is a descriptor of habitat) or a *prospectivity index* (if the points are mineral deposits and the covariate is a geological variable). This command uses a nonparametric smoothing method which does not assume a particular form for the relationship.

If `object` is a point pattern, and `baseline` is missing or null, this command assumes that `object` is a realisation of a Poisson point process with intensity function $\lambda(u)$ of the form

$$\lambda(u) = \rho(Z(u))$$

where Z is the spatial covariate function given by `covariate`, and $\rho(z)$ is a function to be estimated. This command computes estimators of $\rho(z)$ proposed by Baddeley and Turner (2005) and Baddeley et al (2012).

The covariate Z must have continuous values.

If `object` is a point pattern, and `baseline` is given, then the intensity function is assumed to be

$$\lambda(u) = \rho(Z(u))B(u)$$

where $B(u)$ is the baseline intensity at location u. A smoothing estimator of the relative intensity $\rho(z)$ is computed.

If `object` is a fitted point process model, suppose `X` is the original data point pattern to which the model was fitted. Then this command assumes `X` is a realisation of a Poisson point process with intensity function of the form

$$\lambda(u) = \rho(Z(u))\kappa(u)$$

where $\kappa(u)$ is the intensity of the fitted model `object`. A smoothing estimator of $\rho(z)$ is computed.

The estimation procedure is determined by the character strings `method` and `smoother` and the argument `horvitz`. The estimation procedure involves computing several density estimates and combining them. The algorithm used to compute density estimates is determined by `smoother`:

- If `smoother="kernel"`, each the smoothing procedure is based on fixed-bandwidth kernel density estimation, performed by `density.default`.
- If `smoother="local"`, the smoothing procedure is based on local likelihood density estimation, performed by `locfit`.

The `method` determines how the density estimates will be combined to obtain an estimate of $\rho(z)$:

- If `method="ratio"`, then $\rho(z)$ is estimated by the ratio of two density estimates. The numerator is a (rescaled) density estimate obtained by smoothing the values $Z(y_i)$ of the covariate Z observed at the data points y_i. The denominator is a density estimate of the reference distribution of Z.
- If `method="reweight"`, then $\rho(z)$ is estimated by applying density estimation to the values $Z(y_i)$ of the covariate Z observed at the data points y_i, with weights inversely proportional to the reference density of Z.
- If `method="transform"`, the smoothing method is variable-bandwidth kernel smoothing, implemented by applying the Probability Integral Transform to the covariate values, yielding values in the range 0 to 1, then applying edge-corrected density estimation on the interval $[0, 1]$, and back-transforming.

If `horvitz=TRUE`, then the calculations described above are modified by using Horvitz-Thompson weighting. The contribution to the numerator from each data point is weighted by the reciprocal of the baseline value or fitted intensity value at that data point; and a corresponding adjustment is made to the denominator.

The covariate will be evaluated on a fine grid of locations, with spatial resolution controlled by the arguments `dimyx,eps,nd,random`. In two dimensions (i.e. if `object` is of class `"ppp"`, `"ppm"` or `"quad"`) the arguments `dimyx, eps` are passed to `as.mask` to control the pixel resolution. On a linear network (i.e. if `object` is of class `"lpp"`) the argument `nd` specifies the total number of test locations on the linear network, `eps` specifies the linear separation between test locations, and `random` specifies whether the test locations have a randomised starting position.

If the argument `weights` is present, then the contribution from each data point `X[i]` to the estimate of ρ is multiplied by `weights[i]`.

If the argument `subset` is present, then the calculations are performed using only the data inside this spatial region.

Value

A function value table (object of class `"fv"`) containing the estimated values of ρ for a sequence of values of Z. Also belongs to the class `"rhohat"` which has special methods for `print`, `plot` and `predict`.

Categorical and discrete covariates

This technique assumes that the covariate has continuous values. It is not applicable to covariates with categorical (factor) values or discrete values such as small integers. For a categorical covariate, use `intensity.quadratcount` applied to the result of `quadratcount(X, tess=covariate)`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Ya-Mei Chang, Yong Song, and Rolf Turner `<r.turner@auckland.ac.nz>`.

References

Baddeley, A., Chang, Y.-M., Song, Y. and Turner, R. (2012) Nonparametric estimation of the dependence of a point process on spatial covariates. *Statistics and Its Interface* **5** (2), 221–236.

Baddeley, A. and Turner, R. (2005) Modelling spatial point patterns in R. In: A. Baddeley, P. Gregori, J. Mateu, R. Stoica, and D. Stoyan, editors, *Case Studies in Spatial Point Pattern Modelling*, Lecture Notes in Statistics number 185. Pages 23–74. Springer-Verlag, New York, 2006. ISBN: 0-387-28311-0.

See Also

`rho2hat`, `methods.rhohat`, `parres`.

See `ppm` for a parametric method for the same problem.

Examples

```
  X <-  rpoispp(function(x,y){exp(3+3*x)})
  rho <- rhohat(X, "x")
  rho <- rhohat(X, function(x,y){x})
  plot(rho)
  curve(exp(3+3*x), lty=3, col=2, add=TRUE)

  rhoB <- rhohat(X, "x", method="reweight")
  rhoC <- rhohat(X, "x", method="transform")



  fit <- ppm(X, ~x)
  rr <- rhohat(fit, "y")

# linear network
  Y <- runiflpp(30, simplenet)
  rhoY <- rhohat(Y, "y")
```

`ripras` — *Estimate window from points alone*

Description

Given an observed pattern of points, computes the Ripley-Rasson estimate of the spatial domain from which they came.

Usage

```
ripras(x, y=NULL, shape="convex", f)
```

Arguments

x	vector of x coordinates of observed points, or a 2-column matrix giving x,y coordinates, or a list with components x,y giving coordinates (such as a point pattern object of class "ppp".)
y	(optional) vector of y coordinates of observed points, if x is a vector.
shape	String indicating the type of window to be estimated: either "convex" or "rectangle".
f	(optional) scaling factor. See Details.

Details

Given an observed pattern of points with coordinates given by x and y, this function computes an estimate due to Ripley and Rasson (1977) of the spatial domain from which the points came.

The points are assumed to have been generated independently and uniformly distributed inside an unknown domain D.

If shape="convex" (the default), the domain D is assumed to be a convex set. The maximum likelihood estimate of D is the convex hull of the points (computed by convexhull.xy). Analogously to the problems of estimating the endpoint of a uniform distribution, the MLE is not optimal. Ripley and Rasson's estimator is a rescaled copy of the convex hull, centred at the centroid of the convex hull. The scaling factor is $1/sqrt(1-m/n)$ where n is the number of data points and m the number of vertices of the convex hull. The scaling factor may be overridden using the argument f.

If shape="rectangle", the domain D is assumed to be a rectangle with sides parallel to the coordinate axes. The maximum likelihood estimate of D is the bounding box of the points (computed by bounding.box.xy). The Ripley-Rasson estimator is a rescaled copy of the bounding box, with scaling factor $(n+1)/(n-1)$ where n is the number of data points, centred at the centroid of the bounding box. The scaling factor may be overridden using the argument f.

Value

A window (an object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Ripley, B.D. and Rasson, J.-P. (1977) Finding the edge of a Poisson forest. *Journal of Applied Probability*, **14**, 483 – 491.

See Also

owin, as.owin, bounding.box.xy, convexhull.xy

Examples

```
x <- runif(30)
y <- runif(30)
w <- ripras(x,y)
plot(owin(), main="ripras(x,y)")
plot(w, add=TRUE)
points(x,y)

X <- rpoispp(15)
plot(X, main="ripras(X)")
plot(ripras(X), add=TRUE)

# two points insufficient
ripras(c(0,1),c(0,0))
# triangle
ripras(c(0,1,0.5), c(0,0,1))
# three collinear points
ripras(c(0,0,0), c(0,1,2))
```

`rjitter` *Random Perturbation of a Point Pattern*

Description

Applies independent random displacements to each point in a point pattern.

Usage

```
rjitter(X, radius, retry=TRUE, giveup = 10000, ..., nsim=1, drop=TRUE)
```

Arguments

`X`	A point pattern (object of class "ppp").
`radius`	Scale of perturbations. A positive numerical value. The displacement vectors will be uniformly distributed in a circle of this radius. There is a sensible default.
`retry`	What to do when a perturbed point lies outside the window of the original point pattern. If `retry=FALSE`, the point will be lost; if `retry=TRUE`, the algorithm will try again.
`giveup`	Maximum number of unsuccessful attempts.
`...`	Ignored.
`nsim`	Number of simulated realisations to be generated.
`drop`	Logical. If `nsim=1` and `drop=TRUE` (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

Each of the points in the point pattern `X` is subjected to an independent random displacement. The displacement vectors are uniformly distributed in a circle of radius `radius`.

If a displaced point lies outside the window, then if `retry=FALSE` the point will be lost.

However if retry=TRUE, the algorithm will try again: each time a perturbed point lies outside the window, the algorithm will reject it and generate another proposed perturbation of the original point, until one lies inside the window, or until giveup unsuccessful attempts have been made. In the latter case, any unresolved points will be included without any perturbation. The return value will always be a point pattern with the same number of points as X.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1, in the same window as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
X <- rsyst(owin(), 10, 10)
Y <- rjitter(X, 0.02)
plot(Y)
Z <- rjitter(X)
```

---

rknn *Theoretical Distribution of Nearest Neighbour Distance*

Description

Density, distribution function, quantile function and random generation for the random distance to the kth nearest neighbour in a Poisson point process in d dimensions.

Usage

```
dknn(x, k = 1, d = 2, lambda = 1)
pknn(q, k = 1, d = 2, lambda = 1)
qknn(p, k = 1, d = 2, lambda = 1)
rknn(n, k = 1, d = 2, lambda = 1)
```

Arguments

x,q	vector of quantiles.
p	vector of probabilities.
n	number of observations to be generated.
k	order of neighbour.
d	dimension of space.
lambda	intensity of Poisson point process.

Details

In a Poisson point process in d-dimensional space, let the random variable R be the distance from a fixed point to the k-th nearest random point, or the distance from a random point to the k-th nearest other random point.

Then R^d has a Gamma distribution with shape parameter k and rate $\lambda * \alpha$ where α is a constant (equal to the volume of the unit ball in d-dimensional space). See e.g. Cressie (1991, page 61).

These functions support calculation and simulation for the distribution of R.

Value

A numeric vector: dknn returns the probability density, pknn returns cumulative probabilities (distribution function), qknn returns quantiles, and rknn generates random deviates.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Cressie, N.A.C. (1991) *Statistics for spatial data*. John Wiley and Sons, 1991.

Examples

```
x <- seq(0, 5, length=20)
densities <- dknn(x, k=3, d=2)
cdfvalues <- pknn(x, k=3, d=2)
randomvalues <- rknn(100, k=3, d=2)
deciles <- qknn((1:9)/10, k=3, d=2)
```

---

rlabel *Random Re-Labelling of Point Pattern*

Description

Randomly allocates marks to a point pattern, or permutes the existing marks, or resamples from the existing marks.

Usage

```
rlabel(X, labels=marks(X), permute=TRUE)
```

Arguments

X	Point pattern (object of class "ppp", "lpp", "pp3" or "ppx").
labels	Vector of values from which the new marks will be drawn at random. Defaults to the vector of existing marks.
permute	Logical value indicating whether to generate new marks by randomly permuting labels or by drawing a random sample with replacement.

Details

This very simple function allocates random marks to an existing point pattern X. It is useful for hypothesis testing purposes.

In the simplest case, the command rlabel(X) yields a point pattern obtained from X by randomly permuting the marks of the points.

If permute=TRUE, then labels should be a vector of length equal to the number of points in X. The result of rlabel will be a point pattern with locations given by X and marks given by a random permutation of labels (i.e. a random sample without replacement).

If permute=FALSE, then labels may be a vector of any length. The result of rlabel will be a point pattern with locations given by X and marks given by a random sample from labels (with replacement).

Value

A marked point pattern (of the same class as X).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

marks<- to assign arbitrary marks.

Examples

```
data(amacrine)

# Randomly permute the marks "on" and "off"
# Result always has 142 "off" and 152 "on"
Y <- rlabel(amacrine)

# randomly allocate marks "on" and "off"
# with probabilities p(off) = 0.48, p(on) = 0.52
Y <- rlabel(amacrine, permute=FALSE)

# randomly allocate marks "A" and "B" with equal probability
data(cells)
Y <- rlabel(cells, labels=factor(c("A", "B")), permute=FALSE)
```

---

rLGCP *Simulate Log-Gaussian Cox Process*

Description

Generate a random point pattern, a realisation of the log-Gaussian Cox process.

Usage

```
rLGCP(model="exp", mu = 0, param = NULL,
      ...,
      win=NULL, saveLambda=TRUE, nsim=1, drop=TRUE)
```

Arguments

model	character string: the short name of a covariance model for the Gaussian random field. After adding the prefix "RM", the code will search for a function of this name in the **RandomFields** package.
mu	mean function of the Gaussian random field. Either a single number, a function(x,y, ...) or a pixel image (object of class "im").
param	List of parameters for the covariance. Standard arguments are var and scale.
...	Additional parameters for the covariance, or arguments passed to as.mask to determine the pixel resolution.
win	Window in which to simulate the pattern. An object of class "owin".
saveLambda	Logical. If TRUE (the default) then the simulated random intensity will also be saved, and returns as an attribute of the point pattern.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates a realisation of a log-Gaussian Cox process (LGCP). This is a Cox point process in which the logarithm of the random intensity is a Gaussian random field with mean function μ and covariance function $c(r)$. Conditional on the random intensity, the point process is a Poisson process with this intensity.

The string model specifies the covariance function of the Gaussian random field, and the parameters of the covariance are determined by param and ....

To determine the covariance model, the string model is prefixed by "RM", and a function of this name is sought in the **RandomFields** package. For a list of available models see RMmodel in the **RandomFields** package. For example the Matérn covariance is specified by model="matern", corresponding to the function RMmatern in the **RandomFields** package.

Standard variance parameters (for all functions beginning with "RM" in the **RandomFields** package) are var for the variance at distance zero, and scale for the scale parameter. Other parameters are specified in the help files for the individual functions beginning with "RM". For example the help file for RMmatern states that nu is a parameter for this model.

This algorithm uses the function RFsimulate in the **RandomFields** package to generate values of a Gaussian random field, with the specified mean function mu and the covariance specified by the arguments model and param, on the points of a regular grid. The exponential of this random field is taken as the intensity of a Poisson point process, and a realisation of the Poisson process is then generated by the function rpoispp in the **spatstat** package.

If the simulation window win is missing or NULL, then it defaults to Window(mu) if mu is a pixel image, and it defaults to the unit square otherwise.

The LGCP model can be fitted to data using kppm.

Value

A point pattern (object of class "ppp") or a list of point patterns.

Additionally, the simulated intensity function for each point pattern is returned as an attribute "Lambda" of the point pattern, if saveLambda=TRUE.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Modified by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Møller, J., Syversveen, A. and Waagepetersen, R. (1998) Log Gaussian Cox Processes. *Scandinavian Journal of Statistics* **25**, 451–482.

See Also

rpoispp, rMatClust, rGaussPoisson, rNeymanScott, lgcp.estK, kppm

Examples

```
if(require(RandomFields)) {
# homogeneous LGCP with exponential covariance function
X <- rLGCP("exp", 3, var=0.2, scale=.1)

# inhomogeneous LGCP with Gaussian covariance function
m <- as.im(function(x, y){5 - 1.5 * (x - 0.5)^2 + 2 * (y - 0.5)^2}, W=owin())
X <- rLGCP("gauss", m, var=0.15, scale =0.5)
plot(attr(X, "Lambda"))
points(X)

# inhomogeneous LGCP with Matern covariance function
X <- rLGCP("matern", function(x, y){ 1 - 0.4 * x},
           var=2, scale=0.7, nu=0.5,
           win = owin(c(0, 10), c(0, 10)))
plot(X)
}
```

---

rlinegrid *Generate grid of parallel lines with random displacement*

Description

Generates a grid of parallel lines, equally spaced, inside the specified window.

Usage

```
rlinegrid(angle = 45, spacing = 0.1, win = owin())
```

Arguments

angle	Common orientation of the lines, in degrees anticlockwise from the x axis.
spacing	Spacing between successive lines.
win	Window in which to generate the lines. An object of class "owin" or something acceptable to as.owin.

Details

The grid is randomly displaced from the origin.

Value

A line segment pattern (object of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

psp, rpoisline

Examples

```
plot(rlinegrid(30, 0.05))
```

---

rlpp *Random Points on a Linear Network*

Description

Generates n independent random points on a linear network with a specified probability density.

Usage

```
rlpp(n, f, ..., nsim=1, drop=TRUE)
```

Arguments

n	Number of random points to generate. A nonnegative integer giving the number of points, or an integer vector giving the numbers of points of each type.
f	Probability density (not necessarily normalised). A pixel image on a linear network (object of class "linim") or a function on a linear network (object of class "linfun"). Alternatively, f can be a list of functions or pixel images, giving the densities of points of each type.
...	Additional arguments passed to f if it is a function or a list of functions.
nsim	Number of simulated realisations to generate.
drop	Logical value indicating what to do when nsim=1. If drop=TRUE (the default), the result is a point pattern. If drop=FALSE, the result is a list with one entry which is a point pattern.

Details

The linear network L, on which the points will be generated, is determined by the argument f.

If f is a function, it is converted to a pixel image on the linear network, using any additional function arguments ....

If n is a single integer and f is a function or pixel image, then independent random points are generated on L with probability density proportional to f.

If n is an integer vector and f is a list of functions or pixel images, where n and f have the same length, then independent random points of several types are generated on L, with n[i] points of type i having probability density proportional to f[[i]].

Value

If nsim = 1 and drop=TRUE, a point pattern on the linear network, i.e.\ an object of class "lpp". Otherwise, a list of such point patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

runiflpp

Examples

```
g <- function(x, y, seg, tp) { exp(x + 3*y) }
f <- linfun(g, simplenet)

rlpp(20, f)

plot(rlpp(20, f, nsim=3))
```

---

rMatClust *Simulate Matern Cluster Process*

Description

Generate a random point pattern, a simulated realisation of the Matérn Cluster Process.

Usage

```
rMatClust(kappa, scale, mu, win = owin(c(0,1),c(0,1)),
          nsim=1, drop=TRUE,
          saveLambda=FALSE, expand = scale, ...,
          poisthresh=1e-6, saveparents=TRUE)
```

Arguments

kappa	Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.
scale	Radius parameter of the clusters.
mu	Mean number of points per cluster (a single positive number) or reference intensity for the cluster points (a function or a pixel image).
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.
saveLambda	Logical. If TRUE then the random intensity corresponding to the simulated parent points will also be calculated and saved, and returns as an attribute of the point pattern.
expand	Numeric. Size of window expansion for generation of parent points. Defaults to scale which is the cluster radius.
...	Passed to clusterfield to control the image resolution when saveLambda=TRUE.
poisthresh	Numerical threshold below which the model will be treated as a Poisson process. See Details.
saveparents	Logical value indicating whether to save the locations of the parent points as an attribute.

Details

This algorithm generates a realisation of Matérn's cluster process, a special case of the Neyman-Scott process, inside the window win.

In the simplest case, where kappa and mu are single numbers, the algorithm generates a uniform Poisson point process of "parent" points with intensity kappa. Then each parent point is replaced by a random cluster of "offspring" points, the number of points per cluster being Poisson (mu) distributed, and their positions being placed and uniformly inside a disc of radius scale centred on the parent point. The resulting point pattern is a realisation of the classical "stationary Matern cluster process" generated inside the window win. This point process has intensity kappa * mu.

The algorithm can also generate spatially inhomogeneous versions of the Matérn cluster process:

- The parent points can be spatially inhomogeneous. If the argument kappa is a function(x,y) or a pixel image (object of class "im"), then it is taken as specifying the intensity function of an inhomogeneous Poisson process that generates the parent points.
- The offspring points can be inhomogeneous. If the argument mu is a function(x,y) or a pixel image (object of class "im"), then it is interpreted as the reference density for offspring points, in the sense of Waagepetersen (2007). For a given parent point, the offspring constitute a Poisson process with intensity function equal to mu/(pi * scale^2) inside the disc of radius scale centred on the parent point, and zero intensity outside this disc. Equivalently we first generate, for each parent point, a Poisson (M) random number of offspring (where M is the maximum value of mu) placed independently and uniformly in the disc of radius scale centred on the parent location, and then randomly thin the offspring points, with retention probability mu/M.
- Both the parent points and the offspring points can be inhomogeneous, as described above.

Note that if kappa is a pixel image, its domain must be larger than the window win. This is because an offspring point inside win could have its parent point lying outside win. In order to allow this, the simulation algorithm first expands the original window win by a distance expand and generates the Poisson process of parent points on this larger window. If kappa is a pixel image, its domain must contain this larger window.

The intensity of the Matérn cluster process is kappa * mu if either kappa or mu is a single number. In the general case the intensity is an integral involving kappa, mu and scale.

The Matérn cluster process model with homogeneous parents (i.e. where kappa is a single number) can be fitted to data using kppm. Currently it is not possible to fit the Matérn cluster process model with inhomogeneous parents.

If the pair correlation function of the model is very close to that of a Poisson process, deviating by less than poisthresh, then the model is approximately a Poisson process, and will be simulated as a Poisson process with intensity kappa * mu, using rpoispp. This avoids computations that would otherwise require huge amounts of memory.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Additionally, some intermediate results of the simulation are returned as attributes of this point pattern (see rNeymanScott). Furthermore, the simulated intensity function is returned as an attribute "Lambda", if saveLambda=TRUE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Matérn, B. (1960) *Spatial Variation*. Meddelanden från Statens Skogsforskningsinstitut, volume 59, number 5. Statens Skogsforskningsinstitut, Sweden.

Matérn, B. (1986) *Spatial Variation*. Lecture Notes in Statistics 36, Springer-Verlag, New York.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

rpoispp, rThomas, rCauchy, rVarGamma, rNeymanScott, rGaussPoisson, kppm, clusterfit.

Examples

```
# homogeneous
X <- rMatClust(10, 0.05, 4)
# inhomogeneous
ff <- function(x,y){ 4 * exp(2 * abs(x) - 1) }
Z <- as.im(ff, owin())
Y <- rMatClust(10, 0.05, Z)
YY <- rMatClust(ff, 0.05, 3)
```

rMaternI *Simulate Matern Model I*

Description

Generate a random point pattern, a simulated realisation of the Matérn Model I inhibition process model.

Usage

```
rMaternI(kappa, r, win = owin(c(0,1),c(0,1)), stationary=TRUE, ...,
         nsim=1, drop=TRUE)
```

Arguments

kappa	Intensity of the Poisson process of proposal points. A single positive number.
r	Inhibition distance.
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin. Alternatively a higher-dimensional box of class "box3" or "boxx".
stationary	Logical. Whether to start with a stationary process of proposal points (stationary=TRUE) or to generate the proposal points only inside the window (stationary=FALSE).
...	Ignored.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This algorithm generates one or more realisations of Matérn's Model I inhibition process inside the window win.

The process is constructed by first generating a uniform Poisson point process of "proposal" points with intensity kappa. If stationary = TRUE (the default), the proposal points are generated in a window larger than win that effectively means the proposals are stationary. If stationary=FALSE then the proposal points are only generated inside the window win.

A proposal point is then deleted if it lies within r units' distance of another proposal point. Otherwise it is retained.

The retained points constitute Matérn's Model I.

Value

A point pattern if nsim=1, or a list of point patterns if nsim > 1. Each point pattern is normally an object of class "ppp", but may be of class "pp3" or "ppx" depending on the window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Ute Hahn, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

rpoispp, rMatClust

Examples

```
X <- rMaternI(20, 0.05)
Y <- rMaternI(20, 0.05, stationary=FALSE)
```

---

rMaternII *Simulate Matern Model II*

Description

Generate a random point pattern, a simulated realisation of the Matérn Model II inhibition process.

Usage

```
rMaternII(kappa, r, win = owin(c(0,1),c(0,1)), stationary=TRUE, ...,
          nsim=1, drop=TRUE)
```

Arguments

kappa	Intensity of the Poisson process of proposal points. A single positive number.
r	Inhibition distance.
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin. Alternatively a higher-dimensional box of class "box3" or "boxx".
stationary	Logical. Whether to start with a stationary process of proposal points (stationary=TRUE) or to generate the proposal points only inside the window (stationary=FALSE).
...	Ignored.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This algorithm generates one or more realisations of Matérn's Model II inhibition process inside the window win.

The process is constructed by first generating a uniform Poisson point process of "proposal" points with intensity kappa. If stationary = TRUE (the default), the proposal points are generated in a window larger than win that effectively means the proposals are stationary. If stationary=FALSE then the proposal points are only generated inside the window win.

Then each proposal point is marked by an "arrival time", a number uniformly distributed in $[0, 1]$ independently of other variables.

A proposal point is deleted if it lies within r units' distance of another proposal point *that has an earlier arrival time*. Otherwise it is retained. The retained points constitute Matérn's Model II.

The difference between Matérn's Model I and II is the italicised statement above. Model II has a higher intensity for the same parameter values.

Value

A point pattern if nsim=1, or a list of point patterns if nsim > 1. Each point pattern is normally an object of class "ppp", but may be of class "pp3" or "ppx" depending on the window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Ute Hahn, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

rpoispp, rMatClust, rMaternI

Examples

```
X <- rMaternII(20, 0.05)
Y <- rMaternII(20, 0.05, stationary=FALSE)
```

---

rmh *Simulate point patterns using the Metropolis-Hastings algorithm.*

Description

Generic function for running the Metropolis-Hastings algorithm to produce simulated realisations of a point process model.

Usage

```
rmh(model, ...)
```

Arguments

model	The point process model to be simulated.
...	Further arguments controlling the simulation.

Details

The Metropolis-Hastings algorithm can be used to generate simulated realisations from a wide range of spatial point processes. For caveats, see below.

The function rmh is generic; it has methods rmh.ppm (for objects of class "ppm") and rmh.default (the default). The actual implementation of the Metropolis-Hastings algorithm is contained in rmh.default. For details of its use, see rmh.ppm or rmh.default.

[If the model is a Poisson process, then Metropolis-Hastings is not used; the Poisson model is generated directly using rpoispp or rmpoispp.]

In brief, the Metropolis-Hastings algorithm is a Markov Chain, whose states are spatial point patterns, and whose limiting distribution is the desired point process. After running the algorithm for a very large number of iterations, we may regard the state of the algorithm as a realisation from the desired point process.

However, there are difficulties in deciding whether the algorithm has run for “long enough”. The convergence of the algorithm may indeed be extremely slow. No guarantees of convergence are given!

While it is fashionable to decry the Metropolis-Hastings algorithm for its poor convergence and other properties, it has the advantage of being easy to implement for a wide range of models.

Value

A point pattern, in the form of an object of class "ppp". See rmh.default for details.

Warning

As of version 1.22-1 of spatstat a subtle change was made to rmh.default(). We had noticed that the results produced were sometimes not “scalable” in that two models, differing in effect only by the units in which distances are measured and starting from the same seed, gave different results. This was traced to an idiosyncracy of floating point arithmetic. The code of rmh.default() has been changed so that the results produced by rmh are now scalable. The downside of this is that code which users previously ran may now give results which are different from what they formerly were.

In order to recover former behaviour (so that previous results can be reproduced) set spatstat.options(scalable=FALS See the last example in the help for rmh.default.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rmh.default

Examples

```
# See examples in rmh.default and rmh.ppm
```

---

rmh.default *Simulate Point Process Models using the Metropolis-Hastings Algorithm.*

Description

Generates a random point pattern, simulated from a chosen point process model, using the Metropolis-Hastings algorithm.

Usage

```
   ## Default S3 method:
rmh(model, start=NULL,
   control=default.rmhcontrol(model),
   ...,
   nsim=1, drop=TRUE, saveinfo=TRUE,
   verbose=TRUE, snoop=FALSE)
```

Arguments

model	Data specifying the point process model that is to be simulated.
start	Data determining the initial state of the algorithm.
control	Data controlling the iterative behaviour and termination of the algorithm.
...	Further arguments passed to rmhcontrol or to trend functions in model.
nsim	Number of simulated point patterns that should be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a single point pattern.
saveinfo	Logical value indicating whether to save auxiliary information.
verbose	Logical value indicating whether to print progress reports.
snoop	Logical. If TRUE, activate the visual debugger.

Details

This function generates simulated realisations from any of a range of spatial point processes, using the Metropolis-Hastings algorithm. It is the default method for the generic function rmh.

This function executes a Metropolis-Hastings algorithm with birth, death and shift proposals as described in Geyer and Møller (1994).

The argument model specifies the point process model to be simulated. It is either a list, or an object of class "rmhmodel", with the following components:

cif A character string specifying the choice of interpoint interaction for the point process.

par Parameter values for the conditional intensity function.

w (Optional) window in which the pattern is to be generated. An object of class "owin", or data acceptable to as.owin.

trend Data specifying the spatial trend in the model, if it has a trend. This may be a function, a pixel image (of class "im"), (or a list of functions or images if the model is multitype).

If the trend is a function or functions, any auxiliary arguments ... to rmh.default will be passed to these functions, which should be of the form function(x, y, ...).

types List of possible types, for a multitype point process.

For full details of these parameters, see rmhmodel.default.

The argument start determines the initial state of the Metropolis-Hastings algorithm. It is either NULL, or an object of class "rmhstart", or a list with the following components:

n.start Number of points in the initial point pattern. A single integer, or a vector of integers giving the numbers of points of each type in a multitype point pattern. Incompatible with x.start.

x.start Initial point pattern configuration. Incompatible with n.start.

x.start may be a point pattern (an object of class "ppp"), or data which can be coerced to this class by as.ppp, or an object with components x and y, or a two-column matrix. In the last two cases, the window for the pattern is determined by model$w. In the first two cases, if model$w is also present, then the final simulated pattern will be clipped to the window model$w.

For full details of these parameters, see rmhstart.

The third argument control controls the simulation procedure (including *conditional simulation*), iterative behaviour, and termination of the Metropolis-Hastings algorithm. It is either NULL, or a list, or an object of class "rmhcontrol", with components:

p The probability of proposing a "shift" (as opposed to a birth or death) in the Metropolis-Hastings algorithm.

q The conditional probability of proposing a death (rather than a birth) given that birth/death has been chosen over shift.

nrep The number of repetitions or iterations to be made by the Metropolis-Hastings algorithm. It should be large.

expand Either a numerical expansion factor, or a window (object of class `"owin"`). Indicates that the process is to be simulated on a larger domain than the original data window `w`, then clipped to `w` when the algorithm has finished.

The default is to expand the simulation window if the model is stationary and non-Poisson (i.e. it has no trend and the interaction is not Poisson) and not to expand in all other cases.

If the model has a trend, then in order for expansion to be feasible, the trend must be given either as a function, or an image whose bounding box is large enough to contain the expanded window.

periodic A logical scalar; if `periodic` is `TRUE` we simulate a process on the torus formed by identifying opposite edges of a rectangular window.

ptypes A vector of probabilities (summing to 1) to be used in assigning a random type to a new point.

fixall A logical scalar specifying whether to condition on the number of points of each type.

nverb An integer specifying how often "progress reports" (which consist simply of the number of repetitions completed) should be printed out. If nverb is left at 0, the default, the simulation proceeds silently.

x.cond If this argument is present, then *conditional simulation* will be performed, and `x.cond` specifies the conditioning points and the type of conditioning.

nsave,nburn If these values are specified, then intermediate states of the simulation algorithm will be saved every `nsave` iterations, after an initial burn-in period of `nburn` iterations.

track Logical flag indicating whether to save the transition history of the simulations.

For full details of these parameters, see `rmhcontrol`. The control parameters can also be given in the `...` arguments.

Value

A point pattern (an object of class `"ppp"`, see `ppp.object`) or a list of point patterns.

The returned value has an attribute `info` containing modified versions of the arguments `model`, `start`, and `control` which together specify the exact simulation procedure. The `info` attribute can be printed (and is printed automatically by `summary.ppp`). For computational efficiency, the `info` attribute can be omitted by setting `saveinfo=FALSE`.

The value of `.Random.seed` at the start of the simulations is also saved and returned as an attribute `seed`.

If the argument `track=TRUE` was given (see `rmhcontrol`), the transition history of the algorithm is saved, and returned as an attribute `history`. The transition history is a data frame containing a factor `proposaltype` identifying the proposal type (Birth, Death or Shift) and a logical vector `accepted` indicating whether the proposal was accepted. The data frame also has columns `numerator`, `denominator` which give the numerator and denominator of the Hastings ratio for the proposal.

If the argument `nsave` was given (see `rmhcontrol`), the return value has an attribute `saved` which is a list of point patterns, containing the intermediate states of the algorithm.

Conditional Simulation

There are several kinds of conditional simulation.

- Simulation *conditional upon the number of points*, that is, holding the number of points fixed. To do this, set `control$p` (the probability of a shift) equal to 1. The number of points is then determined by the starting state, which may be specified either by setting `start$n.start` to be a scalar, or by setting the initial pattern `start$x.start`.
- In the case of multitype processes, it is possible to simulate the model *conditionally upon the number of points of each type*, i.e. holding the number of points of each type to be fixed. To do this, set `control$p` equal to 1 and `control$fixall` to be `TRUE`. The number of points is then determined by the starting state, which may be specified either by setting `start$n.start` to be an integer vector, or by setting the initial pattern `start$x.start`.
- Simulation *conditional on the configuration observed in a sub-window*, that is, requiring that, inside a specified sub-window V, the simulated pattern should agree with a specified point pattern y.To do this, set `control$x.cond` to equal the specified point pattern y, making sure that it is an object of class `"ppp"` and that the window `Window(control$x.cond)` is the conditioning window V.
- Simulation *conditional on the presence of specified points*, that is, requiring that the simulated pattern should include a specified set of points. This is simulation from the Palm distribution of the point process given a pattern y. To do this, set `control$x.cond` to be a `data.frame` containing the coordinates (and marks, if appropriate) of the specified points.

For further information, see `rmhcontrol`.

Note that, when we simulate conditionally on the number of points, or conditionally on the number of points of each type, no expansion of the window is possible.

Visual Debugger

If `snoop = TRUE`, an interactive debugger is activated. On the current plot device, the debugger displays the current state of the Metropolis-Hastings algorithm together with the proposed transition to the next state. Clicking on this graphical display (using the left mouse button) will re-centre the display at the clicked location. Surrounding this graphical display is an array of boxes representing different actions. Clicking on one of the action boxes (using the left mouse button) will cause the action to be performed. Debugger actions include:

- Zooming in or out
- Panning (shifting the field of view) left, right, up or down
- Jumping to the next iteration
- Skipping 10, 100, 1000, 10000 or 100000 iterations
- Jumping to the next Birth proposal (etc)
- Changing the fate of the proposal (i.e. changing whether the proposal is accepted or rejected)
- Dumping the current state and proposal to a file
- Printing detailed information at the terminal
- Exiting the debugger (so that the simulation algorithm continues without further interruption).

Right-clicking the mouse will also cause the debugger to exit.

Warnings

There is never a guarantee that the Metropolis-Hastings algorithm has converged to its limiting distribution.

If `start$x.start` is specified then `expand` is set equal to 1 and simulation takes place in `Window(x.start)`. Any specified value for `expand` is simply ignored.

The presence of both a component `w` of `model` and a non-null value for `Window(x.start)` makes sense ONLY if `w` is contained in `Window(x.start)`.

For multitype processes make sure that, even if there is to be no trend corresponding to a particular type, there is still a component (a NULL component) for that type, in the list.

Other models

In theory, any finite point process model can be simulated using the Metropolis-Hastings algorithm, provided the conditional intensity is uniformly bounded.

In practice, the list of point process models that can be simulated using `rmh.default` is limited to those that have been implemented in the package's internal C code. More options will be added in the future.

Note that the `lookup` conditional intensity function permits the simulation (in theory, to any desired degree of approximation) of any pairwise interaction process for which the interaction depends only on the distance between the pair of points.

Reproducible simulations

If the user wants the simulation to be exactly reproducible (e.g. for a figure in a journal article, where it is useful to have the figure consistent from draft to draft) then the state of the random number generator should be set before calling `rmh.default`. This can be done either by calling `set.seed` or by assigning a value to `.Random.seed`. In the examples below, we use `set.seed`.

If a simulation has been performed and the user now wants to repeat it exactly, the random seed should be extracted from the simulated point pattern `X` by `seed <- attr(x, "seed")`, then assigned to the system random nunber state by `.Random.seed <- seed` before calling `rmh.default`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283 – 322.

Diggle, P. J. (2003) *Statistical Analysis of Spatial Point Patterns* (2nd ed.) Arnold, London.

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

Diggle, P.J., Gates, D.J., and Stibbard, A. (1987) A nonparametric estimator for pairwise-interaction point processes. Biometrika **74**, 763 – 770.

Geyer, C.J. and Møller, J. (1994) Simulation procedures and likelihood inference for spatial point processes. *Scandinavian Journal of Statistics* **21**, 359–373.

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

rmh, rmh.ppm, rStrauss, ppp, ppm, AreaInter, BadGey, DiggleGatesStibbard, DiggleGratton, Fiksel, Geyer, Hardcore, LennardJones, MultiHard, MultiStrauss, MultiStraussHard, PairPiece, Poisson, Softcore, Strauss, StraussHard, Triplets

Examples

```
if(interactive()) {
  nr   <- 1e5
  nv  <- 5000
  ns <- 200
} else {
  nr  <- 10
  nv <- 5
  ns <- 20
  oldopt <- spatstat.options()
  spatstat.options(expand=1.1)
}
set.seed(961018)

# Strauss process.
mod01 <- list(cif="strauss",par=list(beta=2,gamma=0.2,r=0.7),
              w=c(0,10,0,10))
X1.strauss <- rmh(model=mod01,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))

if(interactive()) plot(X1.strauss)

# Strauss process, conditioning on n = 42:
X2.strauss <- rmh(model=mod01,start=list(n.start=42),
                  control=list(p=1,nrep=nr,nverb=nv))

# Tracking algorithm progress:
X <- rmh(model=mod01,start=list(n.start=ns),
         control=list(nrep=nr, nsave=nr/5, nburn=nr/2, track=TRUE))
History <- attr(X, "history")
Saved <- attr(X, "saved")
head(History)
plot(Saved)

# Hard core process:
mod02 <- list(cif="hardcore",par=list(beta=2,hc=0.7),w=c(0,10,0,10))
X3.hardcore <- rmh(model=mod02,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))

if(interactive()) plot(X3.hardcore)

# Strauss process equal to pure hardcore:
mod02s <- list(cif="strauss",par=list(beta=2,gamma=0,r=0.7),w=c(0,10,0,10))
X3.strauss <- rmh(model=mod02s,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))
```

```
# Strauss process in a polygonal window.
x     <- c(0.55,0.68,0.75,0.58,0.39,0.37,0.19,0.26,0.42)
y     <- c(0.20,0.27,0.68,0.99,0.80,0.61,0.45,0.28,0.33)
mod03 <- list(cif="strauss",par=list(beta=2000,gamma=0.6,r=0.07),
             w=owin(poly=list(x=x,y=y)))
X4.strauss <- rmh(model=mod03,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))
if(interactive()) plot(X4.strauss)

# Strauss process in a polygonal window, conditioning on n = 80.
X5.strauss <- rmh(model=mod03,start=list(n.start=ns),
                  control=list(p=1,nrep=nr,nverb=nv))

# Strauss process, starting off from X4.strauss, but with the
# polygonal window replace by a rectangular one.  At the end,
# the generated pattern is clipped to the original polygonal window.
xxx <- X4.strauss
Window(xxx) <- as.owin(c(0,1,0,1))
X6.strauss <- rmh(model=mod03,start=list(x.start=xxx),
                  control=list(nrep=nr,nverb=nv))

# Strauss with hardcore:
mod04 <- list(cif="straush",par=list(beta=2,gamma=0.2,r=0.7,hc=0.3),
             w=c(0,10,0,10))
X1.straush <- rmh(model=mod04,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))

# Another Strauss with hardcore (with a perhaps surprising result):
mod05 <- list(cif="straush",par=list(beta=80,gamma=0.36,r=45,hc=2.5),
             w=c(0,250,0,250))
X2.straush <- rmh(model=mod05,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))

# Pure hardcore (identical to X3.strauss).
mod06 <- list(cif="straush",par=list(beta=2,gamma=1,r=1,hc=0.7),
             w=c(0,10,0,10))
X3.straush <- rmh(model=mod06,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))

# Soft core:
w    <- c(0,10,0,10)
mod07 <- list(cif="sftcr",par=list(beta=0.8,sigma=0.1,kappa=0.5),
             w=c(0,10,0,10))
X.sftcr <- rmh(model=mod07,start=list(n.start=ns),
               control=list(nrep=nr,nverb=nv))
if(interactive()) plot(X.sftcr)

# Area-interaction process:
mod42 <- rmhmodel(cif="areaint",par=list(beta=2,eta=1.6,r=0.7),
              w=c(0,10,0,10))
X.area <- rmh(model=mod42,start=list(n.start=ns),
               control=list(nrep=nr,nverb=nv))
if(interactive()) plot(X.area)

# Triplets process
modtrip <- list(cif="triplets",par=list(beta=2,gamma=0.2,r=0.7),
```

```
                w=c(0,10,0,10))
X.triplets <- rmh(model=modtrip,
                  start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))
if(interactive()) plot(X.triplets)

# Multitype Strauss:
beta <- c(0.027,0.008)
gmma <- matrix(c(0.43,0.98,0.98,0.36),2,2)
r    <- matrix(c(45,45,45,45),2,2)
mod08 <- list(cif="straussm",par=list(beta=beta,gamma=gmma,radii=r),
             w=c(0,250,0,250))
X1.straussm <- rmh(model=mod08,start=list(n.start=ns),
                   control=list(ptypes=c(0.75,0.25),nrep=nr,nverb=nv))
if(interactive()) plot(X1.straussm)

# Multitype Strauss conditioning upon the total number
# of points being 80:
X2.straussm <- rmh(model=mod08,start=list(n.start=ns),
                   control=list(p=1,ptypes=c(0.75,0.25),nrep=nr,
                                nverb=nv))

# Conditioning upon the number of points of type 1 being 60
# and the number of points of type 2 being 20:
X3.straussm <- rmh(model=mod08,start=list(n.start=c(60,20)),
                   control=list(fixall=TRUE,p=1,ptypes=c(0.75,0.25),
                                nrep=nr,nverb=nv))

# Multitype Strauss hardcore:
rhc  <- matrix(c(9.1,5.0,5.0,2.5),2,2)
mod09 <- list(cif="straushm",par=list(beta=beta,gamma=gmma,
             iradii=r,hradii=rhc),w=c(0,250,0,250))
X.straushm <- rmh(model=mod09,start=list(n.start=ns),
                  control=list(ptypes=c(0.75,0.25),nrep=nr,nverb=nv))

# Multitype Strauss hardcore with trends for each type:
beta  <- c(0.27,0.08)
tr3   <- function(x,y){x <- x/250; y <- y/250;
    exp((6*x + 5*y - 18*x^2 + 12*x*y - 9*y^2)/6)
                      }
                      # log quadratic trend
tr4   <- function(x,y){x <- x/250; y <- y/250;
                      exp(-0.6*x+0.5*y)}
                     # log linear trend
mod10 <- list(cif="straushm",par=list(beta=beta,gamma=gmma,
              iradii=r,hradii=rhc),w=c(0,250,0,250),
              trend=list(tr3,tr4))
X1.straushm.trend <- rmh(model=mod10,start=list(n.start=ns),
                         control=list(ptypes=c(0.75,0.25),
                         nrep=nr,nverb=nv))
if(interactive()) plot(X1.straushm.trend)

# Multitype Strauss hardcore with trends for each type, given as images:
bigwin <- square(250)
i1 <- as.im(tr3, bigwin)
i2 <- as.im(tr4, bigwin)
mod11 <- list(cif="straushm",par=list(beta=beta,gamma=gmma,
```

```
              iradii=r,hradii=rhc),w=bigwin,
              trend=list(i1,i2))
X2.straushm.trend <- rmh(model=mod11,start=list(n.start=ns),
                         control=list(ptypes=c(0.75,0.25),expand=1,
                         nrep=nr,nverb=nv))

# Diggle, Gates, and Stibbard:
mod12 <- list(cif="dgs",par=list(beta=3600,rho=0.08),w=c(0,1,0,1))
X.dgs <- rmh(model=mod12,start=list(n.start=ns),
             control=list(nrep=nr,nverb=nv))
if(interactive()) plot(X.dgs)

# Diggle-Gratton:
mod13 <- list(cif="diggra",
              par=list(beta=1800,kappa=3,delta=0.02,rho=0.04),
              w=square(1))
X.diggra <- rmh(model=mod13,start=list(n.start=ns),
                control=list(nrep=nr,nverb=nv))
if(interactive()) plot(X.diggra)

# Fiksel:
modFik <- list(cif="fiksel",
              par=list(beta=180,r=0.15,hc=0.07,kappa=2,a= -1.0),
              w=square(1))
X.fiksel <- rmh(model=modFik,start=list(n.start=ns),
                control=list(nrep=nr,nverb=nv))
if(interactive()) plot(X.fiksel)

# Geyer:
mod14 <- list(cif="geyer",par=list(beta=1.25,gamma=1.6,r=0.2,sat=4.5),
              w=c(0,10,0,10))
X1.geyer <- rmh(model=mod14,start=list(n.start=ns),
                control=list(nrep=nr,nverb=nv))
if(interactive()) plot(X1.geyer)

# Geyer; same as a Strauss process with parameters
# (beta=2.25,gamma=0.16,r=0.7):

mod15 <- list(cif="geyer",par=list(beta=2.25,gamma=0.4,r=0.7,sat=10000),
              w=c(0,10,0,10))
X2.geyer <- rmh(model=mod15,start=list(n.start=ns),
                control=list(nrep=nr,nverb=nv))

mod16 <- list(cif="geyer",par=list(beta=8.1,gamma=2.2,r=0.08,sat=3))
data(redwood)
X3.geyer <- rmh(model=mod16,start=list(x.start=redwood),
                control=list(periodic=TRUE,nrep=nr,nverb=nv))

# Geyer, starting from the redwood data set, simulating
# on a torus, and conditioning on n:
X4.geyer <- rmh(model=mod16,start=list(x.start=redwood),
                control=list(p=1,periodic=TRUE,nrep=nr,nverb=nv))

# Lookup (interaction function h_2 from page 76, Diggle (2003)):
   r <- seq(from=0,to=0.2,length=101)[-1] # Drop 0.
   h <- 20*(r-0.05)
   h[r<0.05] <- 0
```

```
   h[r>0.10] <- 1
   mod17 <- list(cif="lookup",par=list(beta=4000,h=h,r=r),w=c(0,1,0,1))
   X.lookup <- rmh(model=mod17,start=list(n.start=ns),
                   control=list(nrep=nr,nverb=nv))
   if(interactive()) plot(X.lookup)

# Strauss with trend
tr <- function(x,y){x <- x/250; y <- y/250;
    exp((6*x + 5*y - 18*x^2 + 12*x*y - 9*y^2)/6)
                      }
beta <- 0.3
gmma <- 0.5
r    <- 45
modStr <- list(cif="strauss",par=list(beta=beta,gamma=gmma,r=r),
              w=square(250), trend=tr)
X1.strauss.trend <- rmh(model=modStr,start=list(n.start=ns),
                        control=list(nrep=nr,nverb=nv))
# Baddeley-Geyer
r <- seq(0,0.2,length=8)[-1]
gmma <- c(0.5,0.6,0.7,0.8,0.7,0.6,0.5)
mod18 <- list(cif="badgey",par=list(beta=4000, gamma=gmma,r=r,sat=5),
              w=square(1))
X1.badgey <- rmh(model=mod18,start=list(n.start=ns),
                 control=list(nrep=nr,nverb=nv))
mod19 <- list(cif="badgey",
              par=list(beta=4000, gamma=gmma,r=r,sat=1e4),
              w=square(1))
set.seed(1329)
X2.badgey <- rmh(model=mod18,start=list(n.start=ns),
                 control=list(nrep=nr,nverb=nv))

# Check:
h <- ((prod(gmma)/cumprod(c(1,gmma)))[-8])^2
hs <- stepfun(r,c(h,1))
mod20 <- list(cif="lookup",par=list(beta=4000,h=hs),w=square(1))
set.seed(1329)
X.check <- rmh(model=mod20,start=list(n.start=ns),
                   control=list(nrep=nr,nverb=nv))
# X2.badgey and X.check will be identical.

mod21 <- list(cif="badgey",par=list(beta=300,gamma=c(1,0.4,1),
              r=c(0.035,0.07,0.14),sat=5), w=square(1))
X3.badgey <- rmh(model=mod21,start=list(n.start=ns),
                 control=list(nrep=nr,nverb=nv))
# Same result as Geyer model with beta=300, gamma=0.4, r=0.07,
# sat = 5 (if seeds and control parameters are the same)

# Or more simply:
mod22 <- list(cif="badgey",
              par=list(beta=300,gamma=0.4,r=0.07, sat=5),
              w=square(1))
X4.badgey <- rmh(model=mod22,start=list(n.start=ns),
                 control=list(nrep=nr,nverb=nv))
# Same again --- i.e. the BadGey model includes the Geyer model.


# Illustrating scalability.
```

```
## Not run:
 M1 <- rmhmodel(cif="strauss",par=list(beta=60,gamma=0.5,r=0.04),w=owin())
 set.seed(496)
 X1 <- rmh(model=M1,start=list(n.start=300))
 M2 <- rmhmodel(cif="strauss",par=list(beta=0.6,gamma=0.5,r=0.4),
           w=owin(c(0,10),c(0,10)))
 set.seed(496)
 X2  <- rmh(model=M2,start=list(n.start=300))
 chk <- affine(X1,mat=diag(c(10,10)))
 all.equal(chk,X2,check.attributes=FALSE)
 # Under the default spatstat options the foregoing all.equal()
 # will yield TRUE.  Setting spatstat.options(scalable=FALSE) and
 # re-running the code will reveal differences between X1 and X2.

## End(Not run)

   if(!interactive()) spatstat.options(oldopt)
```

---

rmh.ppm *Simulate from a Fitted Point Process Model*

Description

Given a point process model fitted to data, generate a random simulation of the model, using the Metropolis-Hastings algorithm.

Usage

```
  ## S3 method for class 'ppm'
rmh(model, start=NULL,
                      control=default.rmhcontrol(model, w=w),
                      ...,
                      w = NULL,
                      project=TRUE,
                      nsim=1, drop=TRUE, saveinfo=TRUE,
                      verbose=TRUE, new.coef=NULL)
```

Arguments

`model`	A fitted point process model (object of class "ppm", see `ppm.object`) which it is desired to simulate. This fitted model is usually the result of a call to `ppm`. See **Details** below.
`start`	Data determining the initial state of the Metropolis-Hastings algorithm. See `rmhstart` for description of these arguments. Defaults to `list(x.start=data.ppm(model))`
`control`	Data controlling the iterative behaviour of the Metropolis-Hastings algorithm. See `rmhcontrol` for description of these arguments.
`...`	Further arguments passed to `rmhcontrol`, or to `rmh.default`, or to covariate functions in the model.
`w`	Optional. Window in which the simulations should be generated. Default is the window of the original data.

project	Logical flag indicating what to do if the fitted model is invalid (in the sense that the values of the fitted coefficients do not specify a valid point process). If project=TRUE the closest valid model will be simulated; if project=FALSE an error will occur.
nsim	Number of simulated point patterns that should be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a single point pattern.
saveinfo	Logical value indicating whether to save auxiliary information.
verbose	Logical flag indicating whether to print progress reports.
new.coef	New values for the canonical parameters of the model. A numeric vector of the same length as coef(model).

Details

This function generates simulated realisations from a point process model that has been fitted to point pattern data. It is a method for the generic function rmh for the class "ppm" of fitted point process models. To simulate other kinds of point process models, see rmh or rmh.default.

The argument model describes the fitted model. It must be an object of class "ppm" (see ppm.object), and will typically be the result of a call to the point process model fitting function ppm.

The current implementation enables simulation from any fitted model involving the interactions AreaInter, DiggleGratton, DiggleGatesStibbard, Geyer, Hardcore, MultiStrauss, MultiStraussHard, PairPiece, Poisson, Strauss, StraussHard and Softcore, including nonstationary models. See the examples.

It is also possible to simulate *hybrids* of several such models. See Hybrid and the examples.

It is possible that the fitted coefficients of a point process model may be "illegal", i.e. that there may not exist a mathematically well-defined point process with the given parameter values. For example, a Strauss process with interaction parameter $\gamma > 1$ does not exist, but the model-fitting procedure used in ppm will sometimes produce values of γ greater than 1. In such cases, if project=FALSE then an error will occur, while if project=TRUE then rmh.ppm will find the nearest legal model and simulate this model instead. (The nearest legal model is obtained by projecting the vector of coefficients onto the set of valid coefficient vectors. The result is usually the Poisson process with the same fitted intensity.)

The arguments start and control are lists of parameters determining the initial state and the iterative behaviour, respectively, of the Metropolis-Hastings algorithm.

The argument start is passed directly to rmhstart. See rmhstart for details of the parameters of the initial state, and their default values.

The argument control is first passed to rmhcontrol. Then if any additional arguments ... are given, update.rmhcontrol is called to update the parameter values. See rmhcontrol for details of the iterative behaviour parameters, and default.rmhcontrol for their default values.

Note that if you specify expansion of the simulation window using the parameter expand (so that the model will be simulated on a window larger than the original data window) then the model must be capable of extrapolation to this larger window. This is usually not possible for models which depend on external covariates, because the domain of a covariate image is usually the same as the domain of the fitted model.

After extracting the relevant information from the fitted model object model, rmh.ppm invokes the default rmh algorithm rmh.default, unless the model is Poisson. If the model is Poisson then the Metropolis-Hastings algorithm is not needed, and the model is simulated directly, using one of rpoispp, rmpoispp, rpoint or rmpoint.

See rmh.default for further information about the implementation, or about the Metropolis-Hastings algorithm.

Value

A point pattern (an object of class "ppp"; see ppp.object) or a list of point patterns.

Warnings

See Warnings in rmh.default.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

simulate.ppm, rmh, rmhmodel, rmhcontrol, default.rmhcontrol, update.rmhcontrol, rmhstart, rmh.default, ppp.object, ppm,

Interactions: AreaInter, DiggleGratton, DiggleGatesStibbard, Geyer, Hardcore, Hybrid, MultiStrauss, MultiStraussHard, PairPiece, Poisson, Strauss, StraussHard, Softcore

Examples

```
   live <- interactive()
   op <- spatstat.options()
   spatstat.options(rmh.nrep=1e5)
   Nrep <- 1e5

   X <- swedishpines
   if(live) plot(X, main="Swedish Pines data")

   # Poisson process
   fit <- ppm(X, ~1, Poisson())
   Xsim <- rmh(fit)
   if(live) plot(Xsim, main="simulation from fitted Poisson model")

   # Strauss process
   fit <- ppm(X, ~1, Strauss(r=7))
   Xsim <- rmh(fit)
   if(live) plot(Xsim, main="simulation from fitted Strauss model")

   ## Not run:
     # Strauss process simulated on a larger window
     # then clipped to original window
     Xsim <- rmh(fit, control=list(nrep=Nrep, expand=1.1, periodic=TRUE))
     Xsim <- rmh(fit, nrep=Nrep, expand=2, periodic=TRUE)

## End(Not run)

   ## Not run:
     X <- rSSI(0.05, 100)
     # piecewise-constant pairwise interaction function
     fit <- ppm(X, ~1, PairPiece(seq(0.02, 0.1, by=0.01)))
```

```
     Xsim <- rmh(fit)

## End(Not run)

    # marked point pattern
    Y <- amacrine

   ## Not run:
     # marked Poisson models
     fit <- ppm(Y)
     fit <- ppm(Y,~marks)
     fit <- ppm(Y,~polynom(x,2))
     fit <- ppm(Y,~marks+polynom(x,2))
     fit <- ppm(Y,~marks*polynom(x,y,2))
     Ysim <- rmh(fit)

## End(Not run)

   # multitype Strauss models
   MS <- MultiStrauss(radii=matrix(0.07, ncol=2, nrow=2),
                      types = levels(Y$marks))
   ## Not run:
    fit <- ppm(Y ~marks, MS)
    Ysim <- rmh(fit)

## End(Not run)

   fit <- ppm(Y ~ marks*polynom(x,y,2), MS)
   Ysim <- rmh(fit)
   if(live) plot(Ysim, main="simulation from fitted inhomogeneous Multitype Strauss")

   spatstat.options(op)

  ## Not run:
    # Hybrid model
    fit <- ppm(redwood, ~1, Hybrid(A=Strauss(0.02), B=Geyer(0.1, 2)))
    Y <- rmh(fit)

## End(Not run)
```

---

rmhcontrol *Set Control Parameters for Metropolis-Hastings Algorithm.*

Description

Sets up a list of parameters controlling the iterative behaviour of the Metropolis-Hastings algorithm.

Usage

```
rmhcontrol(...)

## Default S3 method:
rmhcontrol(..., p=0.9, q=0.5, nrep=5e5,
                     expand=NULL, periodic=NULL, ptypes=NULL,
```

```
x.cond=NULL, fixall=FALSE, nverb=0,
nsave=NULL, nburn=nsave, track=FALSE,
pstage=c("block", "start"))
```

Arguments

`...`	Arguments passed to methods.
`p`	Probability of proposing a shift (as against a birth/death).
`q`	Conditional probability of proposing a death given that a birth or death will be proposed.
`nrep`	Total number of steps (proposals) of Metropolis-Hastings algorithm that should be run.
`expand`	Simulation window or expansion rule. Either a window (object of class `"owin"`) or a numerical expansion factor, specifying that simulations are to be performed in a domain other than the original data window, then clipped to the original data window. This argument is passed to `rmhexpand`. A numerical expansion factor can be in several formats: see `rmhexpand`.
`periodic`	Logical value (or `NULL`) indicating whether to simulate "periodically", i.e. identifying opposite edges of the rectangular simulation window. A `NULL` value means "undecided."
`ptypes`	For multitype point processes, the distribution of the mark attached to a new random point (when a birth is proposed)
`x.cond`	Conditioning points for conditional simulation.
`fixall`	(Logical) for multitype point processes, whether to fix the number of points of each type.
`nverb`	Progress reports will be printed every `nverb` iterations
`nsave,nburn`	If these values are specified, then intermediate states of the simulation algorithm will be saved every `nsave` iterations, after an initial burn-in period of `nburn` iterations.
`track`	Logical flag indicating whether to save the transition history of the simulations.
`pstage`	Character string specifying when to generate proposal points. Either `"start"` or `"block"`.

Details

The Metropolis-Hastings algorithm, implemented as `rmh`, generates simulated realisations of point process models. The function `rmhcontrol` sets up a list of parameters which control the iterative behaviour and termination of the Metropolis-Hastings algorithm, for use in a subsequent call to `rmh`. It also checks that the parameters are valid.

(A separate function `rmhstart` determines the initial state of the algorithm, and `rmhmodel` determines the model to be simulated.)

The parameters are as follows:

p The probability of proposing a "shift" (as opposed to a birth or death) in the Metropolis-Hastings algorithm.

If $p = 1$ then the algorithm only alters existing points, so the number of points never changes, i.e. we are simulating conditionally upon the number of points. The number of points is determined by the initial state (specified by `rmhstart`).

If $p = 1$ and fixall=TRUE and the model is a multitype point process model, then the algorithm only shifts the locations of existing points and does not alter their marks (types). This is equivalent to simulating conditionally upon the number of points of each type. These numbers are again specified by the initial state.

If $p = 1$ then no expansion of the simulation window is allowed (see expand below).

The default value of p can be changed by setting the parameter rmh.p in spatstat.options.

q The conditional probability of proposing a death (rather than a birth) given that a shift is not proposed. This is of course ignored if p is equal to 1.

The default value of q can be changed by setting the parameter rmh.q in spatstat.options.

nrep The number of repetitions or iterations to be made by the Metropolis-Hastings algorithm. It should be large.

The default value of nrep can be changed by setting the parameter rmh.nrep in spatstat.options.

expand Either a number or a window (object of class "owin"). Indicates that the process is to be simulated on a domain other than the original data window w, then clipped to w when the algorithm has finished. This would often be done in order to approximate the simulation of a stationary process (Geyer, 1999) or more generally a process existing in the whole plane, rather than just in the window w.

If expand is a window object, it is taken as the larger domain in which simulation is performed.

If expand is numeric, it is interpreted as an expansion factor or expansion distance for determining the simulation domain from the data window. It should be a *named* scalar, such as expand=c(area=2), expand=c(distance=0.1), expand=c(length=1.2). See rmhexpand() for more details. If the name is omitted, it defaults to area.

Expansion is not permitted if the number of points has been fixed by setting p = 1 or if the starting configuration has been specified via the argument x.start in rmhstart.

If expand is NULL, this is interpreted to mean "not yet decided". An expansion rule will be determined at a later stage, using appropriate defaults. See rmhexpand.

periodic A logical value (or NULL) determining whether to simulate "periodically". If periodic is TRUE, and if the simulation window is a rectangle, then the simulation algorithm effectively identifies opposite edges of the rectangle. Points near the right-hand edge of the rectangle are deemed to be close to points near the left-hand edge. Periodic simulation usually gives a better approximation to a stationary point process. For periodic simulation, the simulation window must be a rectangle. (The simulation window is determined by expand as described above.)

The value NULL means 'undecided'. The decision is postponed until rmh is called. Depending on the point process model to be simulated, rmh will then set periodic=TRUE if the simulation window is expanded *and* the expanded simulation window is rectangular; otherwise periodic=FALSE.

Note that periodic=TRUE is only permitted when the simulation window (i.e. the expanded window) is rectangular.

ptypes A vector of probabilities (summing to 1) to be used in assigning a random type to a new point. Defaults to a vector each of whose entries is $1/nt$ where nt is the number of types for the process. Convergence of the simulation algorithm should be improved if ptypes is close to the relative frequencies of the types which will result from the simulation.

x.cond If this argument is given, then *conditional simulation* will be performed, and x.cond specifies the location of the fixed points as well as the type of conditioning. It should be either a point pattern (object of class "ppp") or a list(x,y) or a data.frame. See the section on Conditional Simulation.

fixall A logical scalar specifying whether to condition on the number of points of each type. Meaningful only if a marked process is being simulated, and if $p = 1$. A warning message is given if fixall is set equal to TRUE when it is not meaningful.

nverb An integer specifying how often "progress reports" (which consist simply of the number of repetitions completed) should be printed out. If nverb is left at 0, the default, the simulation proceeds silently.

nsave,nburn If these integers are given, then the current state of the simulation algorithm (i.e. the current random point pattern) will be saved every `nsave` iterations, starting from iteration `nburn`.

track Logical flag indicating whether to save the transition history of the simulations (i.e. information specifying what type of proposal was made, and whether it was accepted or rejected, for each iteration).

pstage Character string specifying the stage of the algorithm at which the randomised proposal points should be generated. If `pstage="start"` or if `nsave=0`, the entire sequence of `nrep` random proposal points is generated at the start of the algorithm. This is the original behaviour of the code, and should be used in order to maintain consistency with older versions of **spatstat**. If `pstage="block"` and `nsave > 0`, then a set of `nsave` random proposal points will be generated before each block of `nsave` iterations. This is much more efficient. The default is `pstage="block"`.

Value

An object of class `"rmhcontrol"`, which is essentially a list of parameter values for the algorithm.

There is a `print` method for this class, which prints a sensible description of the parameters chosen.

Conditional Simulation

For a Gibbs point process X, the Metropolis-Hastings algorithm easily accommodates several kinds of conditional simulation:

conditioning on the total number of points: We fix the total number of points $N(X)$ to be equal to n. We simulate from the conditional distribution of X given $N(X) = n$.

conditioning on the number of points of each type: In a multitype point process, where Y_j denotes the process of points of type j, we fix the number $N(Y_j)$ of points of type j to be equal to n_j, for $j = 1, 2, \ldots, m$. We simulate from the conditional distribution of X given $N(Y_j) = n_j$ for $j = 1, 2, \ldots, m$.

conditioning on the realisation in a subwindow: We require that the point process X should, within a specified sub-window V, coincide with a specified point pattern y. We simulate from the conditional distribution of X given $X \cap V = y$.

Palm conditioning: We require that the point process X include a specified list of points y. We simulate from the point process with probability density $g(x) = cf(x \cup y)$ where f is the probability density of the original process X, and c is a normalising constant.

To achieve each of these types of conditioning we do as follows:

conditioning on the total number of points: Set `p=1`. The number of points is determined by the initial state of the simulation: see `rmhstart`.

conditioning on the number of points of each type: Set `p=1` and `fixall=TRUE`. The number of points of each type is determined by the initial state of the simulation: see `rmhstart`.

conditioning on the realisation in a subwindow: Set `x.cond` to be a point pattern (object of class `"ppp"`). Its window `V=Window(x.cond)` becomes the conditioning subwindow V.

Palm conditioning: Set `x.cond` to be a `list(x,y)` or `data.frame` with two columns containing the coordinates of the points, or a `list(x,y,marks)` or `data.frame` with three columns containing the coordinates and marks of the points.

The arguments `x.cond`, `p` and `fixall` can be combined.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

rmh, rmhmodel, rmhstart, rmhexpand, spatstat.options

Examples

```
# parameters given as named arguments
c1 <- rmhcontrol(p=0.3,periodic=TRUE,nrep=1e6,nverb=1e5)

# parameters given as a list
liz <- list(p=0.9, nrep=1e4)
c2 <- rmhcontrol(liz)

# parameters given in rmhcontrol object
c3 <- rmhcontrol(c1)
```

---

rmhexpand *Specify Simulation Window or Expansion Rule*

Description

Specify a spatial domain in which point process simulations will be performed. Alternatively, specify a rule which will be used to determine the simulation window.

Usage

```
rmhexpand(x = NULL, ..., area = NULL, length = NULL, distance = NULL)
```

Arguments

x	Any kind of data determining the simulation window or the expansion rule. A window (object of class "owin") specifying the simulation window, a numerical value specifying an expansion factor or expansion distance, a list containing one numerical value, an object of class "rmhexpand", or NULL.
...	Ignored.
area	Area expansion factor. Incompatible with other arguments.
length	Length expansion factor. Incompatible with other arguments.
distance	Expansion distance (buffer width). Incompatible with other arguments.

Details

In the Metropolis-Hastings algorithm rmh for simulating spatial point processes, simulations are usually carried out on a spatial domain that is larger than the original window of the point process model, then subsequently clipped to the original window.

The command rmhexpand can be used to specify the simulation window, or to specify a rule which will later be used to determine the simulation window from data.

The arguments are all incompatible: at most one of them should be given.

If the first argument x is given, it may be any of the following:

- a window (object of class "owin") specifying the simulation window.
- an object of class "rmhexpand" specifying the expansion rule.
- a single numerical value, without attributes. This will be interpreted as the value of the argument area.
- either c(area=v) or list(area=v), where v is a single numeric value. This will be interpreted as the value of the argument area.
- either c(length=v) or list(length=v), where v is a single numeric value. This will be interpreted as the value of the argument length.
- either c(distance=v) or list(distance=v), where v is a single numeric value. This will be interpreted as the value of the argument distance.
- NULL, meaning that the expansion rule is not yet determined.

If one of the arguments area, length or distance is given, then the simulation window is determined from the original data window as follows.

area The bounding box of the original data window will be extracted, and the simulation window will be a scalar dilation of this rectangle. The argument area should be a numerical value, greater than or equal to 1. It specifies the area expansion factor, i.e. the ratio of the area of the simulation window to the area of the original point process window's bounding box.

length The bounding box of the original data window will be extracted, and the simulation window will be a scalar dilation of this rectangle. The argument length should be a numerical value, greater than or equal to 1. It specifies the length expansion factor, i.e. the ratio of the width (height) of the simulation window to the width (height) of the original point process window's bounding box.

distance The argument distance should be a numerical value, greater than or equal to 0. It specifies the width of a buffer region around the original data window. If the original data window is a rectangle, then this window is extended by a margin of width equal to distance around all sides of the original rectangle. The result is a rectangle. If the original data window is not a rectangle, then morphological dilation is applied using dilation.owin so that a margin or buffer of width equal to distance is created around all sides of the original window. The result is a non-rectangular window, typically of a different shape.

Value

An object of class "rmhexpand" specifying the expansion rule. There is a print method for this class.

Undetermined expansion

If expand=NULL, this is interpreted to mean that the expansion rule is "not yet decided". Expansion will be decided later, by the simulation algorithm rmh. If the model cannot be expanded (for example if the covariate data in the model are not available on a larger domain) then expansion will not occur. If the model can be expanded, then if the point process model has a finite interaction range r, the default is rmhexpand(distance=2*r), and otherwise rmhexpand(area=2).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

expand.owin to apply the rule to a window.

will.expand to test whether expansion will occur.

rmh, rmhcontrol for background details.

Examples

```
rmhexpand()
rmhexpand(2)
rmhexpand(1)
rmhexpand(length=1.5)
rmhexpand(distance=0.1)
rmhexpand(letterR)
```

---

rmhmodel *Define Point Process Model for Metropolis-Hastings Simulation.*

Description

Builds a description of a point process model for use in simulating the model by the Metropolis-Hastings algorithm.

Usage

```
rmhmodel(...)
```

Arguments

...	Arguments specifying the point process model in some format.

Details

Simulated realisations of many point process models can be generated using the Metropolis-Hastings algorithm rmh. The algorithm requires the model to be specified in a particular format: an object of class "rmhmodel".

The function rmhmodel takes a description of a point process model in some other format, and converts it into an object of class "rmhmodel". It also checks that the parameters of the model are valid.

The function rmhmodel is generic, with methods for

fitted point process models: an object of class "ppm", obtained by a call to the model-fitting function ppm. See rmhmodel.ppm.

lists: a list of parameter values in a certain format. See rmhmodel.list.

default: parameter values specified as separate arguments to .... See rmhmodel.default.

Value

An object of class "rmhmodel", which is essentially a list of parameter values for the model.

There is a print method for this class, which prints a sensible description of the model chosen.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Diggle, P. J. (2003) *Statistical Analysis of Spatial Point Patterns* (2nd ed.) Arnold, London.

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

Diggle, P.J., Gates, D.J., and Stibbard, A. (1987) A nonparametric estimator for pairwise-interaction point processes. Biometrika **74**, 763 – 770. *Scandinavian Journal of Statistics* **21**, 359–373.

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

rmhmodel.ppm, rmhmodel.default, rmhmodel.list, rmh, rmhcontrol, rmhstart, ppm, Strauss, Softcore, StraussHard, Triplets, MultiStrauss, MultiStraussHard, DiggleGratton, PairPiece

---

rmhmodel.default *Build Point Process Model for Metropolis-Hastings Simulation.*

Description

Builds a description of a point process model for use in simulating the model by the Metropolis-Hastings algorithm.

Usage

```
  ## Default S3 method:
rmhmodel(...,
         cif=NULL, par=NULL, w=NULL, trend=NULL, types=NULL)
```

Arguments

`...`	Ignored.
`cif`	Character string specifying the choice of model
`par`	Parameters of the model
`w`	Spatial window in which to simulate
`trend`	Specification of the trend in the model
`types`	A vector of factor levels defining the possible marks, for a multitype process.

Details

The generic function `rmhmodel` takes a description of a point process model in some format, and converts it into an object of class `"rmhmodel"` so that simulations of the model can be generated using the Metropolis-Hastings algorithm `rmh`.

This function `rmhmodel.default` is the default method. It builds a description of the point process model from the simple arguments listed.

The argument `cif` is a character string specifying the choice of interpoint interaction for the point process. The current options are

`'areaint'` Area-interaction process.

`'badgey'` Baddeley-Geyer (hybrid Geyer) process.

`'dgs'` Diggle, Gates and Stibbard (1987) process

`'diggra'` Diggle and Gratton (1984) process

`'fiksel'` Fiksel double exponential process (Fiksel, 1984).

`'geyer'` Saturation process (Geyer, 1999).

`'hardcore'` Hard core process

`'lennard'` Lennard-Jones process

`'lookup'` General isotropic pairwise interaction process, with the interaction function specified via a "lookup table".

`'multihard'` Multitype hardcore process

`'penttinen'` The Penttinen process

`'strauss'` The Strauss process

`'straush'` The Strauss process with hard core

`'sftcr'` The Softcore process

`'straussm'` The multitype Strauss process

`'straushm'` Multitype Strauss process with hard core

`'triplets'` Triplets process (Geyer, 1999).

It is also possible to specify a *hybrid* of these interactions in the sense of Baddeley et al (2013). In this case, `cif` is a character vector containing names from the list above. For example, `cif=c('strauss', 'geyer')` would specify a hybrid of the Strauss and Geyer models.

The argument `par` supplies parameter values appropriate to the conditional intensity function being invoked. For the interactions listed above, these parameters are:

areaint: (Area-interaction process.) A **named** list with components `beta,eta,r` which are respectively the "base" intensity, the scaled interaction parameter and the interaction radius.

badgey: (Baddeley-Geyer process.) A **named** list with components beta (the "base" intensity), gamma (a vector of non-negative interaction parameters), r (a vector of interaction radii, of the same length as gamma, in *increasing* order), and sat (the saturation parameter(s); this may be a scalar, or a vector of the same length as gamma and r; all values should be at least 1). Note that because of the presence of "saturation" the gamma values are permitted to be larger than 1.

dgs: (Diggle, Gates, and Stibbard process. See Diggle, Gates, and Stibbard (1987)) A **named** list with components beta and rho. This process has pairwise interaction function equal to

$$e(t) = \sin^2\left(\frac{\pi t}{2\rho}\right)$$

for $t < \rho$, and equal to 1 for $t \geq \rho$.

diggra: (Diggle-Gratton process. See Diggle and Gratton (1984) and Diggle, Gates and Stibbard (1987).) A **named** list with components beta, kappa, delta and rho. This process has pairwise interaction function $e(t)$ equal to 0 for $t < \delta$, equal to

$$\left(\frac{t-\delta}{\rho-\delta}\right)^\kappa$$

for $\delta \leq t < \rho$, and equal to 1 for $t \geq \rho$. Note that here we use the symbol κ where Diggle, Gates, and Stibbard use β since we reserve the symbol β for an intensity parameter.

fiksel: (Fiksel double exponential process, see Fiksel (1984)) A **named** list with components beta, r, hc, kappa and a. This process has pairwise interaction function $e(t)$ equal to 0 for $t < hc$, equal to

$$\exp(a \exp(-\kappa t))$$

for $hc \leq t < r$, and equal to 1 for $t \geq r$.

geyer: (Geyer's saturation process. See Geyer (1999).) A **named** list with components beta, gamma, r, and sat. The components beta, gamma, r are as for the Strauss model, and sat is the "saturation" parameter. The model is Geyer's "saturation" point process model, a modification of the Strauss process in which we effectively impose an upper limit (sat) on the number of neighbours which will be counted as close to a given point.

Explicitly, a saturation point process with interaction radius r, saturation threshold s, and parameters β and γ, is the point process in which each point x_i in the pattern X contributes a factor

$$\beta\gamma^{\min(s,t(x_i,X))}$$

to the probability density of the point pattern, where $t(x_i, X)$ denotes the number of "r-close neighbours" of x_i in the pattern X.

If the saturation threshold s is infinite, the Geyer process reduces to a Strauss process with interaction parameter γ^2 rather than γ.

hardcore: (Hard core process.) A **named** list with components beta and hc where beta is the base intensity and hc is the hard core distance. This process has pairwise interaction function $e(t)$ equal to 1 if $t > hc$ and 0 if $t <= hc$.

lennard: (Lennard-Jones process.) A **named** list with components sigma and epsilon, where sigma is the characteristic diameter and epsilon is the well depth. See LennardJones for explanation.

multihard: (Multitype hard core process.) A **named** list with components beta and hradii, where beta is a vector of base intensities for each type of point, and hradii is a matrix of hard core radii between each pair of types.

penttinen: (Penttinen process.) A **named** list with components beta,gamma,r which are respectively the "base" intensity, the pairwise interaction parameter, and the disc radius. Note that gamma must be less than or equal to 1. See Penttinen for explanation. (Note that there is also an algorithm for perfect simulation of the Penttinen process, rPenttinen)

strauss: (Strauss process.) A **named** list with components beta,gamma,r which are respectively the "base" intensity, the pairwise interaction parameter and the interaction radius. Note that gamma must be less than or equal to 1. (Note that there is also an algorithm for perfect simulation of the Strauss process, rStrauss)

straush: (Strauss process with hardcore.) A **named** list with entries beta,gamma,r,hc where beta, gamma, and r are as for the Strauss process, and hc is the hardcore radius. Of course hc must be less than r.

sftcr: (Softcore process.) A **named** list with components beta,sigma,kappa. Again beta is a "base" intensity. The pairwise interaction between two points $u \neq v$ is

$$\exp\left\{-\left(\frac{\sigma}{||u-v||}\right)^{2/\kappa}\right\}$$

Note that it is necessary that $0 < \kappa < 1$.

straussm: (Multitype Strauss process.) A **named** list with components

- beta: A vector of "base" intensities, one for each possible type.
- gamma: A **symmetric** matrix of interaction parameters, with γ_{ij} pertaining to the interaction between type i and type j.
- radii: A **symmetric** matrix of interaction radii, with entries r_{ij} pertaining to the interaction between type i and type j.

straushm: (Multitype Strauss process with hardcore.) A **named** list with components beta and gamma as for straussm and **two** "radii" components:

- iradii: the interaction radii
- hradii: the hardcore radii

which are both symmetric matrices of nonnegative numbers. The entries of hradii must be less than the corresponding entries of iradii.

triplets: (Triplets process.) A **named** list with components beta,gamma,r which are respectively the "base" intensity, the triplet interaction parameter and the interaction radius. Note that gamma must be less than or equal to 1.

lookup: (Arbitrary pairwise interaction process with isotropic interaction.) A **named** list with components beta, r, and h, or just with components beta and h.

This model is the pairwise interaction process with an isotropic interaction given by any chosen function H. Each pair of points x_i, x_j in the point pattern contributes a factor $H(d(x_i, x_j))$ to the probability density, where d denotes distance and H is the pair interaction function.

The component beta is a (positive) scalar which determines the "base" intensity of the process.

In this implementation, H must be a step function. It is specified by the user in one of two ways.

- **as a vector of values:** If r is present, then r is assumed to give the locations of jumps in the function H, while the vector h gives the corresponding values of the function.
  Specifically, the interaction function $H(t)$ takes the value h[1] for distances t in the interval [0, r[1]); takes the value h[i] for distances t in the interval [r[i-1], r[i]) where $i = 2, \ldots, n$; and takes the value 1 for $t \geq r[n]$. Here n denotes the length of r.
  The components r and h must be numeric vectors of equal length. The r values must be strictly positive, and sorted in increasing order.

The entries of h must be non-negative. If any entry of h is greater than 1, then the entry h[1] must be 0 (otherwise the specified process is non-existent).
Greatest efficiency is achieved if the values of r are equally spaced.
[**Note:** The usage of r and h has *changed* from the previous usage in **spatstat** versions 1.4-7 to 1.5-1, in which ascending order was not required, and in which the first entry of r had to be 0.]

- **as a stepfun object:** If r is absent, then h must be an object of class "stepfun" specifying a step function. Such objects are created by stepfun.
The stepfun object h must be right-continuous (which is the default using stepfun.)
The values of the step function must all be nonnegative. The values must all be less than 1 unless the function is identically zero on some initial interval $[0, r)$. The rightmost value (the value of h(t) for large t) must be equal to 1.
Greatest efficiency is achieved if the jumps (the "knots" of the step function) are equally spaced.

For a hybrid model, the argument par should be a list, of the same length as cif, such that par[[i]] is a list of the parameters required for the interaction cif[i]. See the Examples.

The optional argument trend determines the spatial trend in the model, if it has one. It should be a function or image (or a list of such, if the model is multitype) to provide the value of the trend at an arbitrary point.

trend given as a function: A trend function may be a function of any number of arguments, but the first two must be the x, y coordinates of a point. Auxiliary arguments may be passed to the trend function at the time of simulation, via the ... argument to rmh.
The function **must** be **vectorized**. That is, it must be capable of accepting vector valued x and y arguments. Put another way, it must be capable of calculating the trend value at a number of points, simultaneously, and should return the **vector** of corresponding trend values.

trend given as an image: An image (see im.object) provides the trend values at a grid of points in the observation window and determines the trend value at other points as the value at the nearest grid point.

Note that the trend or trends must be **non-negative**; no checking is done for this.

The optional argument w specifies the window in which the pattern is to be generated. If specified, it must be in a form which can be coerced to an object of class owin by as.owin.

The optional argument types specifies the possible types in a multitype point process. If the model being simulated is multitype, and types is not specified, then this vector defaults to 1:ntypes where ntypes is the number of types.

Value

An object of class "rmhmodel", which is essentially a list of parameter values for the model.

There is a print method for this class, which prints a sensible description of the model chosen.

Warnings in Respect of "lookup"

For the lookup cif, the entries of the r component of par must be *strictly positive* and sorted into ascending order.

Note that if you specify the lookup pairwise interaction function via stepfun() the arguments x and y which are passed to stepfun() are slightly different from r and h: length(y) is equal to 1+length(x); the final entry of y must be equal to 1 — i.e. this value is explicitly supplied by the user rather than getting tacked on internally.

The step function returned by stepfun() must be right continuous (this is the default behaviour of stepfun()) otherwise an error is given.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Turner, R., Mateu, J. and Bevan, A. (2013) Hybrids of Gibbs point process models and their implementation. *Journal of Statistical Software* **55**:11, 1–43. http://www.jstatsoft.org/v55/i11/

Diggle, P. J. (2003) *Statistical Analysis of Spatial Point Patterns* (2nd ed.) Arnold, London.

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

Diggle, P.J., Gates, D.J., and Stibbard, A. (1987) A nonparametric estimator for pairwise-interaction point processes. Biometrika **74**, 763 – 770. *Scandinavian Journal of Statistics* **21**, 359–373.

Fiksel, T. (1984) Estimation of parameterized pair potentials of marked and non-marked Gibbsian point processes. *Electronische Informationsverabeitung und Kybernetika* **20**, 270–278.

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

rmh, rmhcontrol, rmhstart, ppm, AreaInter, BadGey, DiggleGatesStibbard, DiggleGratton, Fiksel, Geyer, Hardcore, Hybrid, LennardJones, MultiStrauss, MultiStraussHard, PairPiece, Penttinen, Poisson, Softcore, Strauss, StraussHard and Triplets.

Examples

```
# Strauss process:
mod01 <- rmhmodel(cif="strauss",par=list(beta=2,gamma=0.2,r=0.7),
              w=c(0,10,0,10))
# The above could also be simulated using 'rStrauss'

# Strauss with hardcore:
mod04 <- rmhmodel(cif="straush",par=list(beta=2,gamma=0.2,r=0.7,hc=0.3),
             w=owin(c(0,10),c(0,5)))

# Hard core:
mod05 <- rmhmodel(cif="hardcore",par=list(beta=2,hc=0.3),
           w=square(5))

# Soft core:
w    <- square(10)
mod07 <- rmhmodel(cif="sftcr",
                  par=list(beta=0.8,sigma=0.1,kappa=0.5),
                  w=w)

# Area-interaction process:
mod42 <- rmhmodel(cif="areaint",par=list(beta=2,eta=1.6,r=0.7),
              w=c(0,10,0,10))

# Baddeley-Geyer process:
mod99 <- rmhmodel(cif="badgey",par=list(beta=0.3,
```

```
                  gamma=c(0.2,1.8,2.4),r=c(0.035,0.07,0.14),sat=5),
                  w=unit.square())

 # Multitype Strauss:
 beta <- c(0.027,0.008)
 gmma <- matrix(c(0.43,0.98,0.98,0.36),2,2)
 r    <- matrix(c(45,45,45,45),2,2)
 mod08 <- rmhmodel(cif="straussm",
                  par=list(beta=beta,gamma=gmma,radii=r),
                  w=square(250))
 # specify types
 mod09 <- rmhmodel(cif="straussm",
                  par=list(beta=beta,gamma=gmma,radii=r),
                  w=square(250),
                  types=c("A", "B"))

 # Multitype Hardcore:
 rhc  <- matrix(c(9.1,5.0,5.0,2.5),2,2)
 mod08hard <- rmhmodel(cif="multihard",
                  par=list(beta=beta,hradii=rhc),
                  w=square(250),
                  types=c("A", "B"))


 # Multitype Strauss hardcore with trends for each type:
 beta  <- c(0.27,0.08)
 ri    <- matrix(c(45,45,45,45),2,2)
 rhc  <- matrix(c(9.1,5.0,5.0,2.5),2,2)
 tr3   <- function(x,y){x <- x/250; y <- y/250;
     exp((6*x + 5*y - 18*x^2 + 12*x*y - 9*y^2)/6)
                      }
                      # log quadratic trend
 tr4   <- function(x,y){x <- x/250; y <- y/250;
                      exp(-0.6*x+0.5*y)}
                     # log linear trend
 mod10 <- rmhmodel(cif="straushm",par=list(beta=beta,gamma=gmma,
              iradii=ri,hradii=rhc),w=c(0,250,0,250),
              trend=list(tr3,tr4))

 # Triplets process:
 mod11 <- rmhmodel(cif="triplets",par=list(beta=2,gamma=0.2,r=0.7),
              w=c(0,10,0,10))

 # Lookup (interaction function h_2 from page 76, Diggle (2003)):
    r <- seq(from=0,to=0.2,length=101)[-1] # Drop 0.
    h <- 20*(r-0.05)
    h[r<0.05] <- 0
    h[r>0.10] <- 1
    mod17 <- rmhmodel(cif="lookup",par=list(beta=4000,h=h,r=r),w=c(0,1,0,1))

# hybrid model
modhy <- rmhmodel(cif=c('strauss', 'geyer'),
                  par=list(list(beta=100,gamma=0.5,r=0.05),
                           list(beta=1, gamma=0.7,r=0.1, sat=2)),
                  w=square(1))
```

rmhmodel.list   *Define Point Process Model for Metropolis-Hastings Simulation.*

Description

Given a list of parameters, builds a description of a point process model for use in simulating the model by the Metropolis-Hastings algorithm.

Usage

```
## S3 method for class 'list'
rmhmodel(model, ...)
```

Arguments

model	A list of parameters. See Details.
...	Optional list of additional named parameters.

Details

The generic function rmhmodel takes a description of a point process model in some format, and converts it into an object of class "rmhmodel" so that simulations of the model can be generated using the Metropolis-Hastings algorithm rmh.

This function rmhmodel.list is the method for lists. The argument model should be a named list of parameters of the form

list(cif, par, w, trend, types)

where cif and par are required and the others are optional. For details about these components, see rmhmodel.default.

The subsequent arguments ... (if any) may also have these names, and they will take precedence over elements of the list model.

Value

An object of class "rmhmodel", which is essentially a validated list of parameter values for the model.

There is a print method for this class, which prints a sensible description of the model chosen.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Diggle, P. J. (2003) *Statistical Analysis of Spatial Point Patterns* (2nd ed.) Arnold, London.

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

Diggle, P.J., Gates, D.J., and Stibbard, A. (1987) A nonparametric estimator for pairwise-interaction point processes. Biometrika **74**, 763 – 770. *Scandinavian Journal of Statistics* **21**, 359–373.

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

rmhmodel, rmhmodel.default, rmhmodel.ppm, rmh, rmhcontrol, rmhstart, ppm, Strauss, Softcore, StraussHard, MultiStrauss, MultiStraussHard, DiggleGratton, PairPiece

Examples

```
# Strauss process:
mod01 <- list(cif="strauss",par=list(beta=2,gamma=0.2,r=0.7),
              w=c(0,10,0,10))
mod01 <- rmhmodel(mod01)

# Strauss with hardcore:
mod04 <- list(cif="straush",par=list(beta=2,gamma=0.2,r=0.7,hc=0.3),
             w=owin(c(0,10),c(0,5)))
mod04 <- rmhmodel(mod04)

# Soft core:
w    <- square(10)
mod07 <- list(cif="sftcr",
                  par=list(beta=0.8,sigma=0.1,kappa=0.5),
                  w=w)
mod07 <- rmhmodel(mod07)

# Multitype Strauss:
beta <- c(0.027,0.008)
gmma <- matrix(c(0.43,0.98,0.98,0.36),2,2)
r    <- matrix(c(45,45,45,45),2,2)
mod08 <- list(cif="straussm",
                  par=list(beta=beta,gamma=gmma,radii=r),
                  w=square(250))
mod08 <- rmhmodel(mod08)

# specify types
mod09 <- rmhmodel(list(cif="straussm",
                  par=list(beta=beta,gamma=gmma,radii=r),
                  w=square(250),
                  types=c("A", "B")))

# Multitype Strauss hardcore with trends for each type:
beta  <- c(0.27,0.08)
ri    <- matrix(c(45,45,45,45),2,2)
rhc  <- matrix(c(9.1,5.0,5.0,2.5),2,2)
tr3   <- function(x,y){x <- x/250; y <- y/250;
    exp((6*x + 5*y - 18*x^2 + 12*x*y - 9*y^2)/6)
                      }
                      # log quadratic trend
tr4   <- function(x,y){x <- x/250; y <- y/250;
                      exp(-0.6*x+0.5*y)}
                     # log linear trend
mod10 <- list(cif="straushm",par=list(beta=beta,gamma=gmma,
              iradii=ri,hradii=rhc),w=c(0,250,0,250),
```

```
                trend=list(tr3,tr4))
mod10 <- rmhmodel(mod10)

# Lookup (interaction function h_2 from page 76, Diggle (2003)):
r <- seq(from=0,to=0.2,length=101)[-1] # Drop 0.
h <- 20*(r-0.05)
h[r<0.05] <- 0
h[r>0.10] <- 1
mod17 <- list(cif="lookup",par=list(beta=4000,h=h,r=r),w=c(0,1,0,1))
mod17 <- rmhmodel(mod17)
```

`rmhmodel.ppm` *Interpret Fitted Model for Metropolis-Hastings Simulation.*

Description

Converts a fitted point process model into a format that can be used to simulate the model by the Metropolis-Hastings algorithm.

Usage

```
  ## S3 method for class 'ppm'
rmhmodel(model, w, ..., verbose=TRUE, project=TRUE,
                         control=rmhcontrol(),
                         new.coef=NULL)
```

Arguments

`model`	Fitted point process model (object of class "ppm").
`w`	Optional. Window in which the simulations should be generated.
`...`	Ignored.
`verbose`	Logical flag indicating whether to print progress reports while the model is being converted.
`project`	Logical flag indicating what to do if the fitted model does not correspond to a valid point process. See Details.
`control`	Parameters determining the iterative behaviour of the simulation algorithm. Passed to `rmhcontrol`.
`new.coef`	New values for the canonical parameters of the model. A numeric vector of the same length as `coef(model)`.

Details

The generic function `rmhmodel` takes a description of a point process model in some format, and converts it into an object of class "rmhmodel" so that simulations of the model can be generated using the Metropolis-Hastings algorithm `rmh`.

This function `rmhmodel.ppm` is the method for the class "ppm" of fitted point process models.

The argument `model` should be a fitted point process model (object of class "ppm") typically obtained from the model-fitting function `ppm`. This will be converted into an object of class "rmhmodel".

The optional argument `w` specifies the window in which the pattern is to be generated. If specified, it must be in a form which can be coerced to an object of class `owin` by `as.owin`.

Not all fitted point process models obtained from ppm can be simulated. We have not yet implemented simulation code for the LennardJones and OrdThresh models.

It is also possible that a fitted point process model obtained from ppm may not correspond to a valid point process. For example a fitted model with the Strauss interpoint interaction may have any value of the interaction parameter γ; however the Strauss process is not well-defined for $\gamma > 1$ (Kelly and Ripley, 1976).

The argument `project` determines what to do in such cases. If `project=FALSE`, a fatal error will occur. If `project=TRUE`, the fitted model parameters will be adjusted to the nearest values which do correspond to a valid point process. For example a Strauss process with $\gamma > 1$ will be projected to a Strauss process with $\gamma = 1$, equivalent to a Poisson process.

Value

An object of class `"rmhmodel"`, which is essentially a list of parameter values for the model.

There is a `print` method for this class, which prints a sensible description of the model chosen.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Diggle, P. J. (2003) *Statistical Analysis of Spatial Point Patterns* (2nd ed.) Arnold, London.

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

Kelly, F.P. and Ripley, B.D. (1976) On Strauss's model for clustering. *Biometrika* **63**, 357–360.

See Also

rmhmodel, rmhmodel.list, rmhmodel.default, rmh, rmhcontrol, rmhstart, ppm, AreaInter, BadGey, DiggleGatesStibbard, DiggleGratton, Fiksel, Geyer, Hardcore, Hybrid, LennardJones, MultiStrauss, MultiStraussHard, PairPiece, Penttinen, Poisson, Softcore, Strauss, StraussHard and Triplets.

Examples

```
fit1 <- ppm(cells ~1, Strauss(0.07))
mod1 <- rmhmodel(fit1)

fit2 <- ppm(cells ~x, Geyer(0.07, 2))
mod2 <- rmhmodel(fit2)

fit3 <- ppm(cells ~x, Hardcore(0.07))
mod3 <- rmhmodel(fit3)

# Then rmh(mod1), etc
```

rmhstart *Determine Initial State for Metropolis-Hastings Simulation.*

Description

Builds a description of the initial state for the Metropolis-Hastings algorithm.

Usage

```
rmhstart(start, ...)
## Default S3 method:
rmhstart(start=NULL, ..., n.start=NULL, x.start=NULL)
```

Arguments

start	An existing description of the initial state in some format. Incompatible with the arguments listed below.
...	There should be no other arguments.
n.start	Number of initial points (to be randomly generated). Incompatible with x.start.
x.start	Initial point pattern configuration. Incompatible with n.start.

Details

Simulated realisations of many point process models can be generated using the Metropolis-Hastings algorithm implemented in rmh.

This function rmhstart creates a full description of the initial state of the Metropolis-Hastings algorithm, *including possibly the initial state of the random number generator*, for use in a subsequent call to rmh. It also checks that the initial state is valid.

The initial state should be specified **either** by the first argument start **or** by the other arguments n.start, x.start etc.

If start is a list, then it should have components named n.start or x.start, with the same interpretation as described below.

The arguments are:

n.start The number of “initial” points to be randomly (uniformly) generated in the simulation window w. Incompatible with x.start.

For a multitype point process, n.start may be a vector (of length equal to the number of types) giving the number of points of each type to be generated.

If expansion of the simulation window is selected (see the argument expand to rmhcontrol), then the actual number of starting points in the simulation will be n.start multiplied by the expansion factor (ratio of the areas of the expanded window and original window).

For faster convergence of the Metropolis-Hastings algorithm, the value of n.start should be roughly equal to (an educated guess at) the expected number of points for the point process inside the window.

x.start Initial point pattern configuration. Incompatible with n.start.

x.start may be a point pattern (an object of class ppp), or an object which can be coerced to this class by as.ppp, or a dataset containing vectors x and y.

If x.start is specified, then expansion of the simulation window (the argument expand of rmhcontrol) is not permitted.

The parameters n.start and x.start are *incompatible*.

Value

An object of class "rmhstart", which is essentially a list of parameters describing the initial point pattern and (optionally) the initial state of the random number generator.

There is a print method for this class, which prints a sensible description of the initial state.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rmh, rmhcontrol, rmhmodel

Examples

```
# 30 random points
a <- rmhstart(n.start=30)

# a particular point pattern
data(cells)
b <- rmhstart(x.start=cells)
```

---

rMosaicField *Mosaic Random Field*

Description

Generate a realisation of a random field which is piecewise constant on the tiles of a given tessellation.

Usage

```
rMosaicField(X,
    rgen = function(n) { sample(0:1, n, replace = TRUE)},
    ...,
    rgenargs=NULL)
```

Arguments

X	A tessellation (object of class "tess").
...	Arguments passed to as.mask determining the pixel resolution.
rgen	Function that generates random values for the tiles of the tessellation.
rgenargs	List containing extra arguments that should be passed to rgen (typically specifying parameters of the distribution of the values).

Details

This function generates a realisation of a random field which is piecewise constant on the tiles of the given tessellation X. The values in each tile are independent and identically distributed.

Value

A pixel image (object of class "im").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rpoislinetess, rMosaicSet

Examples

```
X <- rpoislinetess(3)
plot(rMosaicField(X, runif))
plot(rMosaicField(X, runif, dimyx=256))
plot(rMosaicField(X, rnorm, rgenargs=list(mean=10, sd=2)))

plot(rMosaicField(dirichlet(runifpoint(30)), rnorm))
```

---

rMosaicSet *Mosaic Random Set*

Description

Generate a random set by taking a random selection of tiles of a given tessellation.

Usage

```
rMosaicSet(X, p=0.5)
```

Arguments

X	A tessellation (object of class "tess").
p	Probability of including a given tile. A number strictly between 0 and 1.

Details

Given a tessellation X, this function randomly selects some of the tiles of X, including each tile with probability p independently of the other tiles. The selected tiles are then combined to form a set in the plane.

One application of this is Switzer's (1965) example of a random set which has a Markov property. It is constructed by generating X according to a Poisson line tessellation (see rpoislinetess).

Value

A window (object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

References

Switzer, P. A random set process in the plane with a Markovian property. *Annals of Mathematical Statistics* **36** (1965) 1859–1863.

See Also

rpoislinetess, rMosaicField

Examples

```
# Switzer's random set
X <- rpoislinetess(3)
plot(rMosaicSet(X, 0.5), col="green", border=NA)

# another example
plot(rMosaicSet(dirichlet(runifpoint(30)), 0.4))
```

---

rmpoint *Generate N Random Multitype Points*

Description

Generate a random multitype point pattern with a fixed number of points, or a fixed number of points of each type.

Usage

```
rmpoint(n, f=1, fmax=NULL, win=unit.square(),
        types, ptypes,
        ..., giveup=1000, verbose=FALSE,
        nsim=1, drop=TRUE)
```

Arguments

n	Number of marked points to generate. Either a single number specifying the total number of points, or a vector specifying the number of points of each type.
f	The probability density of the multitype points, usually un-normalised. Either a constant, a vector, a function `f(x,y,m, ...)`, a pixel image, a list of functions `f(x,y,...)` or a list of pixel images.
fmax	An upper bound on the values of `f`. If missing, this number will be estimated.
win	Window in which to simulate the pattern. Ignored if `f` is a pixel image or list of pixel images.
types	All the possible types for the multitype pattern.
ptypes	Optional vector of probabilities for each type.
...	Arguments passed to `f` if it is a function.

giveup	Number of attempts in the rejection method after which the algorithm should stop trying to generate new points.
verbose	Flag indicating whether to report details of performance of the simulation algorithm.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates random multitype point patterns consisting of a fixed number of points.

Three different models are available:

I. Random location and type: If n is a single number and the argument ptypes is missing, then n independent, identically distributed random multitype points are generated. Their locations (x[i],y[i]) and types m[i] have joint probability density proportional to $f(x, y, m)$.

II. Random type, and random location given type: If n is a single number and ptypes is given, then n independent, identically distributed random multitype points are generated. Their types m[i] have probability distribution ptypes. Given the types, the locations (x[i],y[i]) have conditional probability density proportional to $f(x, y, m)$.

III. Fixed types, and random location given type: If n is a vector, then we generate n[i] independent, identically distributed random points of type types[i]. For points of type m the conditional probability density of location (x, y) is proportional to $f(x, y, m)$.

Note that the density f is normalised in different ways in Model I and Models II and III. In Model I the normalised joint density is $g(x, y, m) = f(x, y, m)/Z$ where

$$Z = \sum_m \int \int \lambda(x, y, m) \mathrm{d}x \, \mathrm{d}y$$

while in Models II and III the normalised conditional density is $g(x, y \mid m) = f(x, y, m)/Z_m$ where

$$Z_m = \int \int \lambda(x, y, m) \mathrm{d}x \, \mathrm{d}y.$$

In Model I, the marginal distribution of types is $p_m = Z_m/Z$.

The unnormalised density f may be specified in any of the following ways.

single number: If f is a single number, the conditional density of location given type is uniform. That is, the points of each type are uniformly distributed. In Model I, the marginal distribution of types is also uniform (all possible types have equal probability).

vector: If f is a numeric vector, the conditional density of location given type is uniform. That is, the points of each type are uniformly distributed. In Model I, the marginal distribution of types is proportional to the vector f. In Model II, the marginal distribution of types is ptypes, that is, the values in f are ignored. The argument types defaults to names(f), or if that is null, 1:length(f).

function: If f is a function, it will be called in the form f(x,y,m,...) at spatial location (x,y) for points of type m. In Model I, the joint probability density of location and type is proportional to f(x,y,m,...). In Models II and III, the conditional probability density of location (x,y) given type m is proportional to f(x,y,m,...). The function f must work correctly with vectors x, y and m, returning a vector of function values. (Note that m will be a factor with levels types.) The value fmax must be given and must be an upper bound on the values of f(x,y,m,...) for all locations (x, y) inside the window win and all types m. The argument types must be given.

list of functions: If f is a list of functions, then the functions will be called in the form f[[i]](x,y,...) at spatial location (x,y) for points of type types[i]. In Model I, the joint probability density of location and type is proportional to f[[m]](x,y,...). In Models II and III, the conditional probability density of location (x,y) given type m is proportional to f[[m]](x,y,...). The function f[[i]] must work correctly with vectors x and y, returning a vector of function values. The value fmax must be given and must be an upper bound on the values of f[[i]](x,y,...) for all locations (x, y) inside the window win. The argument types defaults to names(f), or if that is null, 1:length(f).

pixel image: If f is a pixel image object of class "im" (see im.object), the unnormalised density at a location (x,y) for points of any type is equal to the pixel value of f for the pixel nearest to (x,y). In Model I, the marginal distribution of types is uniform. The argument win is ignored; the window of the pixel image is used instead. The argument types must be given.

list of pixel images: If f is a list of pixel images, then the image f[[i]] determines the density values of points of type types[i]. The argument win is ignored; the window of the pixel image is used instead. The argument types defaults to names(f), or if that is null, 1:length(f).

The implementation uses the rejection method. For Model I, rmpoispp is called repeatedly until n points have been generated. It gives up after giveup calls if there are still fewer than n points. For Model II, the types are first generated according to ptypes, then the locations of the points of each type are generated using rpoint. For Model III, the locations of the points of each type are generated using rpoint.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppp.object, owin.object

Examples

```
abc <- c("a","b","c")

##### Model I

rmpoint(25, types=abc)
rmpoint(25, 1, types=abc)
# 25 points, equal probability for each type, uniformly distributed locations

rmpoint(25, function(x,y,m) {rep(1, length(x))}, types=abc)
# same as above
rmpoint(25, list(function(x,y){rep(1, length(x))},
                 function(x,y){rep(1, length(x))},
                 function(x,y){rep(1, length(x))}),
             types=abc)
# same as above
```

```
rmpoint(25, function(x,y,m) { x }, types=abc)
# 25 points, equal probability for each type,
# locations nonuniform with density proportional to x

rmpoint(25, function(x,y,m) { ifelse(m == "a", 1, x) }, types=abc)
rmpoint(25, list(function(x,y) { rep(1, length(x)) },
                function(x,y) { x },
                function(x,y) { x }),
                types=abc)
# 25 points, UNEQUAL probabilities for each type,
# type "a" points uniformly distributed,
# type "b" and "c" points nonuniformly distributed.

##### Model II

rmpoint(25, 1, types=abc, ptypes=rep(1,3)/3)
rmpoint(25, 1, types=abc, ptypes=rep(1,3))
# 25 points, equal probability for each type,
# uniformly distributed locations

rmpoint(25, function(x,y,m) {rep(1, length(x))}, types=abc, ptypes=rep(1,3))
# same as above
rmpoint(25, list(function(x,y){rep(1, length(x))},
                 function(x,y){rep(1, length(x))},
                 function(x,y){rep(1, length(x))}),
             types=abc, ptypes=rep(1,3))
# same as above

rmpoint(25, function(x,y,m) { x }, types=abc, ptypes=rep(1,3))
# 25 points, equal probability for each type,
# locations nonuniform with density proportional to x

rmpoint(25, function(x,y,m) { ifelse(m == "a", 1, x) }, types=abc, ptypes=rep(1,3))
# 25 points, EQUAL probabilities for each type,
# type "a" points uniformly distributed,
# type "b" and "c" points nonuniformly distributed.

###### Model III

rmpoint(c(12, 8, 4), 1, types=abc)
# 12 points of type "a",
# 8 points of type "b",
# 4 points of type "c",
# each uniformly distributed

rmpoint(c(12, 8, 4), function(x,y,m) { ifelse(m=="a", 1, x)}, types=abc)
rmpoint(c(12, 8, 4), list(function(x,y) { rep(1, length(x)) },
                               function(x,y) { x },
                               function(x,y) { x }),
             types=abc)

# 12 points of type "a", uniformly distributed
# 8 points of type "b", nonuniform
# 4 points of type "c", nonuniform


#########
```

```
## Randomising an existing point pattern:
# same numbers of points of each type, uniform random locations (Model III)
rmpoint(table(marks(demopat)), 1, win=Window(demopat))

# same total number of points, distribution of types estimated from X,
# uniform random locations (Model II)
rmpoint(npoints(demopat), 1, types=levels(marks(demopat)), win=Window(demopat),
       ptypes=table(marks(demopat)))
```

---

rmpoispp *Generate Multitype Poisson Point Pattern*

Description

Generate a random point pattern, a realisation of the (homogeneous or inhomogeneous) multitype Poisson process.

Usage

```
rmpoispp(lambda, lmax=NULL, win, types, ...,
         nsim=1, drop=TRUE, warnwin=!missing(win))
```

Arguments

lambda	Intensity of the multitype Poisson process. Either a single positive number, a vector, a `function(x,y,m, ...)`, a pixel image, a list of functions `function(x,y, ...)`, or a list of pixel images.
lmax	An upper bound for the value of `lambda`. May be omitted
win	Window in which to simulate the pattern. An object of class `"owin"` or something acceptable to `as.owin`. Ignored if `lambda` is a pixel image or list of images.
types	All the possible types for the multitype pattern.
...	Arguments passed to `lambda` if it is a function.
nsim	Number of simulated realisations to be generated.
drop	Logical. If `nsim=1` and `drop=TRUE` (the default), the result will be a point pattern, rather than a list containing a point pattern.
warnwin	Logical value specifying whether to issue a warning when `win` is ignored.

Details

This function generates a realisation of the marked Poisson point process with intensity `lambda`.

Note that the intensity function $\lambda(x, y, m)$ is the average number of points **of type m** per unit area near the location (x, y). Thus a marked point process with a constant intensity of 10 and three possible types will have an average of 30 points per unit area, with 10 points of each type on average.

The intensity function may be specified in any of the following ways.

single number: If lambda is a single number, then this algorithm generates a realisation of the uniform marked Poisson process inside the window win with intensity lambda for each type. The total intensity of points of all types is lambda * length(types). The argument types must be given and determines the possible types in the multitype pattern.

vector: If lambda is a numeric vector, then this algorithm generates a realisation of the stationary marked Poisson process inside the window win with intensity lambda[i] for points of type types[i]. The total intensity of points of all types is sum(lambda). The argument types defaults to names(lambda), or if that is null, 1:length(lambda).

function: If lambda is a function, the process has intensity lambda(x,y,m,...) at spatial location (x,y) for points of type m. The function lambda must work correctly with vectors x, y and m, returning a vector of function values. (Note that m will be a factor with levels equal to types.) The value lmax, if present, must be an upper bound on the values of lambda(x,y,m,...) for all locations (x, y) inside the window win and all types m. The argument types must be given.

list of functions: If lambda is a list of functions, the process has intensity lambda[[i]](x,y,...) at spatial location (x,y) for points of type types[i]. The function lambda[[i]] must work correctly with vectors x and y, returning a vector of function values. The value lmax, if given, must be an upper bound on the values of lambda(x,y,...) for all locations (x, y) inside the window win. The argument types defaults to names(lambda), or if that is null, 1:length(lambda).

pixel image: If lambda is a pixel image object of class "im" (see im.object), the intensity at a location (x,y) for points of any type is equal to the pixel value of lambda for the pixel nearest to (x,y). The argument win is ignored; the window of the pixel image is used instead. The argument types must be given.

list of pixel images: If lambda is a list of pixel images, then the image lambda[[i]] determines the intensity of points of type types[i]. The argument win is ignored; the window of the pixel image is used instead. The argument types defaults to names(lambda), or if that is null, 1:length(lambda).

If lmax is missing, an approximate upper bound will be calculated.

To generate an inhomogeneous Poisson process the algorithm uses "thinning": it first generates a uniform Poisson process of intensity lmax for points of each type m, then randomly deletes or retains each point independently, with retention probability $p(x, y, m) = \lambda(x, y, m)/\text{lmax}$.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1. Each point pattern is multitype (it carries a vector of marks which is a factor).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rpoispp for unmarked Poisson point process; rmpoint for a fixed number of random marked points; ppp.object, owin.object.

Examples

```
# uniform bivariate Poisson process with total intensity 100 in unit square
pp <- rmpoispp(50, types=c("a","b"))

# stationary bivariate Poisson process with intensity A = 30, B = 70
pp <- rmpoispp(c(30,70), types=c("A","B"))
pp <- rmpoispp(c(30,70))

# works in any window
data(letterR)
pp <- rmpoispp(c(30,70), win=letterR, types=c("A","B"))

# inhomogeneous lambda(x,y,m)
# note argument 'm' is a factor
lam <- function(x,y,m) { 50 * (x^2 + y^3) * ifelse(m=="A", 2, 1)}
pp <- rmpoispp(lam, win=letterR, types=c("A","B"))
# extra arguments
lam <- function(x,y,m,scal) { scal * (x^2 + y^3) * ifelse(m=="A", 2, 1)}
pp <- rmpoispp(lam, win=letterR, types=c("A","B"), scal=50)

# list of functions lambda[[i]](x,y)
lams <- list(function(x,y){50 * x^2}, function(x,y){20 * abs(y)})
pp <- rmpoispp(lams, win=letterR, types=c("A","B"))
pp <- rmpoispp(lams, win=letterR)
# functions with extra arguments
lams <- list(function(x,y,scal){5 * scal * x^2},
             function(x,y, scal){2 * scal * abs(y)})
pp <- rmpoispp(lams, win=letterR, types=c("A","B"), scal=10)
pp <- rmpoispp(lams, win=letterR, scal=10)

# florid example
lams <- list(function(x,y){
      100*exp((6*x + 5*y - 18*x^2 + 12*x*y - 9*y^2)/6)
                       }
                       # log quadratic trend
             ,
             function(x,y){
                           100*exp(-0.6*x+0.5*y)
                       }
                      # log linear trend
             )
 X <- rmpoispp(lams, win=unit.square(), types=c("on", "off"))

# pixel image
Z <- as.im(function(x,y){30 * (x^2 + y^3)}, letterR)
pp <- rmpoispp(Z, types=c("A","B"))

# list of pixel images
ZZ <- list(
        as.im(function(x,y){20 * (x^2 + y^3)}, letterR),
        as.im(function(x,y){40 * (x^3 + y^2)}, letterR))
pp <- rmpoispp(ZZ, types=c("A","B"))
pp <- rmpoispp(ZZ)

# randomising an existing point pattern
rmpoispp(intensity(amacrine), win=Window(amacrine))
```

rNeymanScott	*Simulate Neyman-Scott Process*

Description

Generate a random point pattern, a realisation of the Neyman-Scott cluster process.

Usage

```
rNeymanScott(kappa, expand, rcluster, win = owin(c(0,1),c(0,1)),
             ..., lmax=NULL, nsim=1, drop=TRUE,
             nonempty=TRUE, saveparents=TRUE)
```

Arguments

kappa	Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.
expand	Size of the expansion of the simulation window for generating parent points. A single non-negative number.
rcluster	A function which generates random clusters, or other data specifying the random cluster mechanism. See Details.
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin.
...	Arguments passed to rcluster.
lmax	Optional. Upper bound on the values of kappa when kappa is a function or pixel image.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.
nonempty	Logical. If TRUE (the default), a more efficient algorithm is used, in which parents are generated conditionally on having at least one offspring point. If FALSE, parents are generated even if they have no offspring. Both choices are valid; the default is recommended unless you need to simulate all the parent points for some other purpose.
saveparents	Logical value indicating whether to save the locations of the parent points as an attribute.

Details

This algorithm generates a realisation of the general Neyman-Scott process, with the cluster mechanism given by the function rcluster.

First, the algorithm generates a Poisson point process of "parent" points with intensity kappa in an expanded window as explained below. Here kappa may be a single positive number, a function kappa(x,y), or a pixel image object of class "im" (see im.object). See rpoispp for details.

Second, each parent point is replaced by a random cluster of points. These clusters are combined together to yield a single point pattern, and the restriction of this pattern to the window win is then returned as the result of rNeymanScott.

The expanded window consists of as.rectangle(win) extended by the amount expand in each direction. The size of the expansion is saved in the attribute "expand" and may be extracted by attr(X, "expand") where X is the generated point pattern.

The argument rcluster specifies the cluster mechanism. It may be either:

- A function which will be called to generate each random cluster (the offspring points of each parent point). The function should expect to be called in the form rcluster(x0,y0,...) for a parent point at a location (x0,y0). The return value of rcluster should specify the coordinates of the points in the cluster; it may be a list containing elements x,y, or a point pattern (object of class "ppp"). If it is a marked point pattern then the result of rNeymanScott will be a marked point pattern.
- A list(mu, f) where mu specifies the mean number of offspring points in each cluster, and f generates the random displacements (vectors pointing from the parent to the offspring). In this case, the number of offspring in a cluster is assumed to have a Poisson distribution, implying that the Neyman-Scott process is also a Cox process. The first element mu should be either a single nonnegative number (interpreted as the mean of the Poisson distribution of cluster size) or a pixel image or a function(x,y) giving a spatially varying mean cluster size (interpreted in the sense of Waagepetersen, 2007). The second element f should be a function that will be called once in the form f(n) to generate n independent and identically distributed displacement vectors (i.e. as if there were a cluster of size n with a parent at the origin (0,0)). The function should return a point pattern (object of class "ppp") or something acceptable to xy.coords that specifies the coordinates of n points.

If required, the intermediate stages of the simulation (the parents and the individual clusters) can also be extracted from the return value of rNeymanScott through the attributes "parents" and "parentid". The attribute "parents" is the point pattern of parent points. The attribute "parentid" is an integer vector specifying the parent for each of the points in the simulated pattern.

Neyman-Scott models where kappa is a single number and rcluster = list(mu,f) can be fitted to data using the function kppm.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Additionally, some intermediate results of the simulation are returned as attributes of this point pattern: see Details.

Inhomogeneous Neyman-Scott Processes

There are several different ways of specifying a spatially inhomogeneous Neyman-Scott process:

- The point process of parent points can be inhomogeneous. If the argument kappa is a function(x,y) or a pixel image (object of class "im"), then it is taken as specifying the intensity function of an inhomogeneous Poisson process according to which the parent points are generated.
- The number of points in a typical cluster can be spatially varying. If the argument rcluster is a list of two elements mu, f and the first entry mu is a function(x,y) or a pixel image (object of class "im"), then mu is interpreted as the reference intensity for offspring points, in the sense of Waagepetersen (2007). For a given parent point, the offspring constitute a Poisson process with intensity function equal to mu(x, y) * g(x-x0, y-y0) where g is the probability density of the offspring displacements generated by the function f.

  Equivalently, clusters are first generated with a constant expected number of points per cluster: the constant is mumax, the maximum of mu. Then the offspring are randomly *thinned* (see rthin) with spatially-varying retention probabilities given by mu/mumax.

- The entire mechanism for generating a cluster can be dependent on the location of the parent point. If the argument `rcluster` is a function, then the cluster associated with a parent point at location `(x0,y0)` will be generated by calling `rcluster(x0, y0, ...)`. The behaviour of this function could depend on the location `(x0,y0)` in any fashion.

Note that if `kappa` is an image, the spatial domain covered by this image must be large enough to include the *expanded* window in which the parent points are to be generated. This requirement means that `win` must be small enough so that the expansion of `as.rectangle(win)` is contained in the spatial domain of `kappa`. As a result, one may wind up having to simulate the process in a window smaller than what is really desired.

In the first two cases, the intensity of the Neyman-Scott process is equal to `kappa * mu` if at least one of `kappa` or `mu` is a single number, and is otherwise equal to an integral involving `kappa`, `mu` and `f`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Neyman, J. and Scott, E.L. (1958) A statistical approach to problems of cosmology. *Journal of the Royal Statistical Society, Series B* **20**, 1–43.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

rpoispp, rThomas, rGaussPoisson, rMatClust, rCauchy, rVarGamma

Examples

```
# each cluster consist of 10 points in a disc of radius 0.2
nclust <- function(x0, y0, radius, n) {
            return(runifdisc(n, radius, centre=c(x0, y0)))
          }
plot(rNeymanScott(10, 0.2, nclust, radius=0.2, n=5))

# multitype Neyman-Scott process (each cluster is a multitype process)
nclust2 <- function(x0, y0, radius, n, types=c("a", "b")) {
   X <- runifdisc(n, radius, centre=c(x0, y0))
   M <- sample(types, n, replace=TRUE)
   marks(X) <- M
   return(X)
}
plot(rNeymanScott(15,0.1,nclust2, radius=0.1, n=5))
```

rnoise *Random Pixel Noise*

Description

Generate a pixel image whose pixel values are random numbers following a specified probability distribution.

Usage

```
rnoise(rgen = runif, w = square(1), ...)
```

Arguments

rgen	Random generator for the pixel values. A function in the R language.
w	Window (region or pixel raster) in which to generate the image. Any data acceptable to as.mask.
...	Arguments, matched by name, to be passed to rgen to specify the parameters of the probability distribution, or passed to as.mask to control the pixel resolution.

Details

The argument w could be a window (class "owin"), a pixel image (class "im") or other data. It is first converted to a binary mask by as.mask using any relevant arguments in ....

Then each pixel inside the window (i.e. with logical value TRUE in the mask) is assigned a random numerical value by calling the function rgen.

The function rgen would typically be one of the standard random variable generators like runif (uniformly distributed random values) or rnorm (Gaussian random values). Its first argument n is the number of values to be generated. Other arguments to rgen must be matched by name.

Value

A pixel image (object of class "im").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

as.mask, as.im, Distributions.

Examples

```
plot(rnoise(), main="Uniform noise")
plot(rnoise(rnorm, dimyx=32, mean=2, sd=1),
     main="White noise")
```

roc *Receiver Operating Characteristic*

Description

Computes the Receiver Operating Characteristic curve for a point pattern or a fitted point process model.

Usage

```
roc(X, ...)

## S3 method for class 'ppp'
roc(X, covariate, ..., high = TRUE)

## S3 method for class 'ppm'
roc(X, ...)

## S3 method for class 'kppm'
roc(X, ...)

## S3 method for class 'lpp'
roc(X, covariate, ..., high = TRUE)

## S3 method for class 'lppm'
roc(X, ...)
```

Arguments

`X`	Point pattern (object of class "ppp" or "lpp") or fitted point process model (object of class "ppm" or "kppm" or "lppm").
`covariate`	Spatial covariate. Either a `function(x,y)`, a pixel image (object of class "im"), or one of the strings "x" or "y" indicating the Cartesian coordinates.
`...`	Arguments passed to `as.mask` controlling the pixel resolution for calculations.
`high`	Logical value indicating whether the threshold operation should favour high or low values of the covariate.

Details

This command computes Receiver Operating Characteristic curve. The area under the ROC is computed by `auc`.

For a point pattern `X` and a covariate `Z`, the ROC is a plot showing the ability of the covariate to separate the spatial domain into areas of high and low density of points. For each possible threshold z, the algorithm calculates the fraction $a(z)$ of area in the study region where the covariate takes a value greater than z, and the fraction $b(z)$ of data points for which the covariate value is greater than z. The ROC is a plot of $b(z)$ against $a(z)$ for all thresholds z.

For a fitted point process model, the ROC shows the ability of the fitted model intensity to separate the spatial domain into areas of high and low density of points. The ROC is **not** a diagnostic for the goodness-of-fit of the model (Lobo et al, 2007).

Value

Function value table (object of class "fv") which can be plotted to show the ROC curve.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Lobo, J.M., Jiménez-Valverde, A. and Real, R. (2007) AUC: a misleading measure of the performance of predictive distribution models. *Global Ecology and Biogeography* **17**(2) 145–151.

Nam, B.-H. and D'Agostino, R. (2002) Discrimination index, the area under the ROC curve. Pages 267–279 in Huber-Carol, C., Balakrishnan, N., Nikulin, M.S. and Mesbah, M., *Goodness-of-fit tests and model validity*, Birkhäuser, Basel.

See Also

auc

Examples

```
plot(roc(swedishpines, "x"))
fit <- ppm(swedishpines ~ x+y)
plot(roc(fit))
```

---

rose *Rose Diagram*

Description

Plots a rose diagram (rose of directions), the analogue of a histogram or density plot for angular data.

Usage

```
rose(x, ...)

## Default S3 method:
rose(x, breaks = NULL, ...,
                        weights=NULL,
                        nclass = NULL,
                        unit = c("degree", "radian"),
                        start=0, clockwise=FALSE,
                        main)

## S3 method for class 'histogram'
rose(x, ...,
                        unit = c("degree", "radian"),
                        start=0, clockwise=FALSE,
```

```
                         main, labels=TRUE, at=NULL, do.plot = TRUE)

## S3 method for class 'density'
rose(x, ...,
                   unit = c("degree", "radian"),
                   start=0, clockwise=FALSE,
                   main, labels=TRUE, at=NULL, do.plot = TRUE)

## S3 method for class 'fv'
rose(x, ...,
                   unit = c("degree", "radian"),
                   start=0, clockwise=FALSE,
                   main, labels=TRUE, at=NULL, do.plot = TRUE)
```

Arguments

`x`	Data to be plotted. A numeric vector containing angles, or a `histogram` object containing a histogram of angular values, or a `density` object containing a smooth density estimate for angular data, or an `fv` object giving a function of an angular argument.
`breaks, nclass`	Arguments passed to `hist` to determine the histogram breakpoints.
`...`	Additional arguments passed to `polygon` controlling the appearance of the plot (or passed from `rose.default` to `hist` to control the calculation of the histogram).
`unit`	The unit in which the angles are expressed.
`start`	The starting direction for measurement of angles, that is, the spatial direction which corresponds to a measured angle of zero. Either a character string giving a compass direction (`"N"` for north, `"S"` for south, `"E"` for east, or `"W"` for west) or a number giving the angle from the the horizontal (East) axis to the starting direction. For example, if `unit="degree"` and `clockwise=FALSE`, then `start=90` and `start="N"` are equivalent. The default is to measure angles anti-clockwise from the horizontal axis (East direction).
`clockwise`	Logical value indicating whether angles increase in the clockwise direction (`clockwise=TRUE`) or anti-clockwise, counter-clockwise direction (`clockwise=FALSE`, the default).
`weights`	Optional vector of numeric weights associated with `x`.
`main`	Optional main title for the plot.
`labels`	Either a logical value indicating whether to plot labels next to the tick marks, or a vector of labels for the tick marks.
`at`	Optional vector of angles at which tick marks should be plotted. Set `at=numeric(0)` to suppress tick marks.
`do.plot`	Logical value indicating whether to really perform the plot.

Details

A rose diagram or rose of directions is the analogue of a histogram or bar chart for data which represent angles in two dimensions. The bars of the bar chart are replaced by circular sectors in the rose diagram.

The function `rose` is generic, with a default method for numeric data, and methods for histograms and function tables.

If x is a numeric vector, it must contain angular values in the range 0 to 360 (if unit="degree") or in the range 0 to 2 * pi (if unit="radian"). A histogram of the data will first be computed using hist. Then the rose diagram of this histogram will be plotted by rose.histogram.

If x is an object of class "histogram" produced by the function hist, representing the histogram of angular data, then the rose diagram of the densities (rather than the counts) in this histogram object will be plotted.

If x is an object of class "density" produced by circdensity or density.default, representing a kernel smoothed density estimate of angular data, then the rose diagram of the density estimate will be plotted.

If x is a function value table (object of class "fv") then the argument of the function will be interpreted as an angle, and the value of the function will be interpreted as the radius.

By default, angles are interpreted using the mathematical convention where the zero angle is the horizontal x axis, and angles increase anti-clockwise. Other conventions can be specified using the arguments start and clockwise. Standard compass directions are obtained by setting unit="degree", start="N" and clockwise=TRUE.

Value

A window (class "owin") containing the plotted region.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

fv, hist, circdensity, density.default.

Examples

```
ang <- runif(1000, max=360)
rose(ang, col="grey")
rose(ang, col="grey", start="N", clockwise=TRUE)
```

---

rotate *Rotate*

Description

Applies a rotation to any two-dimensional object, such as a point pattern or a window.

Usage

```
rotate(X, ...)
```

Arguments

X	Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), or a window (object of class "owin").
...	Data specifying the rotation.

Details

This is generic. Methods are provided for point patterns (rotate.ppp) and windows (rotate.owin).

Value

Another object of the same type, representing the result of rotating X through the specified angle.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rotate.ppp, rotate.owin

---

rotate.im *Rotate a Pixel Image*

Description

Rotates a pixel image

Usage

```
## S3 method for class 'im'
rotate(X, angle=pi/2, ..., centre=NULL)
```

Arguments

X	A pixel image (object of class "im").
angle	Angle of rotation, in radians.
...	Ignored.
centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).

Details

The image is rotated by the angle specified. Angles are measured in radians, anticlockwise. The default is to rotate the image 90 degrees anticlockwise.

Value

Another object of class "im" representing the rotated pixel image.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

affine.im, shift.im, rotate

Examples

```
  Z <- distmap(letterR)
  X <- rotate(Z)
  ## Not run:
  plot(X)

## End(Not run)
  Y <- rotate(X, centre="midpoint")
```

---

rotate.infline *Rotate or Shift Infinite Lines*

Description

Given the coordinates of one or more infinite straight lines in the plane, apply a rotation or shift.

Usage

```
## S3 method for class 'infline'
rotate(X, angle = pi/2, ...)

## S3 method for class 'infline'
shift(X, vec = c(0,0), ...)

## S3 method for class 'infline'
reflect(X)

## S3 method for class 'infline'
flipxy(X)
```

Arguments

X	Object of class "infline" representing one or more infinite straight lines in the plane.
angle	Angle of rotation, in radians.
vec	Translation (shift) vector: a numeric vector of length 2, or a list(x,y), or a point pattern containing one point.
...	Ignored.

Details

These functions are methods for the generic shift, rotate, reflect and flipxy for the class "infline".

An object of class "infline" represents one or more infinite lines in the plane.

Value

Another "infline" object representing the result of the transformation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

infline

Examples

```
L <- infline(v=0.5)

plot(square(c(-1,1)), main="rotate lines", type="n")
points(0, 0, pch=3)
plot(L, col="green")
plot(rotate(L, pi/12), col="red")
plot(rotate(L, pi/6), col="red")
plot(rotate(L, pi/4), col="red")

L <- infline(p=c(0.4, 0.9), theta=pi* c(0.2, 0.6))

plot(square(c(-1,1)), main="shift lines", type="n")
L <- infline(p=c(0.7, 0.8), theta=pi* c(0.2, 0.6))
plot(L, col="green")
plot(shift(L, c(-0.5, -0.4)), col="red")

plot(square(c(-1,1)), main="reflect lines", type="n")
points(0, 0, pch=3)
L <- infline(p=c(0.7, 0.8), theta=pi* c(0.2, 0.6))
plot(L, col="green")
plot(reflect(L), col="red")
```

---

rotate.owin *Rotate a Window*

Description

Rotates a window

Usage

```
## S3 method for class 'owin'
rotate(X, angle=pi/2, ..., rescue=TRUE, centre=NULL)
```

Arguments

X	A window (object of class "owin").
angle	Angle of rotation.
rescue	Logical. If TRUE, the rotated window will be processed by rescue.rectangle.
...	Optional arguments passed to as.mask controlling the resolution of the rotated window, if X is a binary pixel mask. Ignored if X is not a binary mask.
centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).

Details

Rotates the window by the specified angle. Angles are measured in radians, anticlockwise. The default is to rotate the window 90 degrees anticlockwise. The centre of rotation is the origin, by default, unless centre is specified.

Value

Another object of class "owin" representing the rotated window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin.object

Examples

```
  w <- owin(c(0,1),c(0,1))
  v <- rotate(w, pi/3)
  e <- rotate(w, pi/2, centre="midpoint")
  ## Not run:
  plot(v)

## End(Not run)
  w <- as.mask(letterR)
  v <- rotate(w, pi/5)
```

---

rotate.ppp *Rotate a Point Pattern*

Description

Rotates a point pattern

Usage

```
 ## S3 method for class 'ppp'
rotate(X, angle=pi/2, ..., centre=NULL)
```

Arguments

X	A point pattern (object of class "ppp").
angle	Angle of rotation.
...	Arguments passed to rotate.owin affecting the handling of the observation window, if it is a binary pixel mask.
centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).

Details

The points of the pattern, and the window of observation, are rotated about the origin by the angle specified. Angles are measured in radians, anticlockwise. The default is to rotate the pattern 90 degrees anticlockwise. If the points carry marks, these are preserved.

Value

Another object of class "ppp" representing the rotated point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppp.object, rotate.owin

Examples

```
  data(cells)
  X <- rotate(cells, pi/3)
  ## Not run:
  plot(X)

## End(Not run)
```

---

rotate.psp *Rotate a Line Segment Pattern*

Description

Rotates a line segment pattern

Usage

```
 ## S3 method for class 'psp'
rotate(X, angle=pi/2, ..., centre=NULL)
```

Arguments

X	A line segment pattern (object of class "psp").
angle	Angle of rotation.
...	Arguments passed to rotate.owin affecting the handling of the observation window, if it is a binary pixel mask.
centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).

Details

The line segments of the pattern, and the window of observation, are rotated about the origin by the angle specified. Angles are measured in radians, anticlockwise. The default is to rotate the pattern 90 degrees anticlockwise. If the line segments carry marks, these are preserved.

Value

Another object of class "psp" representing the rotated line segment pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

psp.object, rotate.owin, rotate.ppp

Examples

```
oldpar <- par(mfrow=c(2,1))
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(X, main="original")
Y <- rotate(X, pi/4)
plot(Y, main="rotated")
par(oldpar)
```

---

rotmean *Rotational Average of a Pixel Image*

Description

Compute the average pixel value over all rotations of the image about the origin, as a function of distance from the origin.

Usage

```
rotmean(X, ..., origin, padzero=TRUE, Xname, result=c("fv", "im"))
```

Arguments

X	A pixel image.
...	Ignored.
origin	Optional. Origin about which the rotations should be performed. Either a numeric vector or a character string as described in the help for `shift.owin`.
padzero	Logical. If `TRUE` (the default), the value of `X` is assumed to be zero outside the window of `X`. If `FALSE`, the value of `X` is taken to be undefined outside the window of `X`.
Xname	Optional name for `X` to be used in the function labels.
result	Character string specifying the kind of result required: either a function object or a pixel image.

Details

This command computes, for each possible distance r, the average pixel value of the pixels lying at distance r from the origin. Kernel smoothing is used to obtain a smooth function of r.

If `result="fv"` (the default) the result is a function object of class `"fv"` giving the mean pixel value of `X` as a function of distance from the origin.

If `result="im"` the result is a pixel image, with the same dimensions as `X`, giving the mean value of `X` over all pixels lying at the same distance from the origin as the current pixel.

If `padzero=TRUE` (the default), the value of `X` is assumed to be zero outside the window of `X`. The rotational mean at a given distance r is the average value of the image `X` over the *entire* circle of radius r, including zero values outside the window if the circle lies partly outside the window.

If `padzero=FALSE`, the value of `X` is taken to be undefined outside the window of `X`. The rotational mean is the average of the `X` values over the *subset* of the circle of radius r that lies entirely inside the window.

Value

An object of class `"fv"` or `"im"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
if(interactive()) {
  Z <- setcov(square(1))
  plot(rotmean(Z))
  plot(rotmean(Z, result="im"))
} else {
  Z <- setcov(square(1), dimyx=32)
  f <- rotmean(Z)
}
```

round.ppp *Apply Numerical Rounding to Spatial Coordinates*

Description

Apply numerical rounding to the spatial coordinates of a point pattern.

Usage

```
## S3 method for class 'ppp'
round(x, digits = 0)

## S3 method for class 'pp3'
round(x, digits = 0)

## S3 method for class 'ppx'
round(x, digits = 0)
```

Arguments

x	A spatial point pattern in any dimension (object of class "ppp", "pp3" or "ppx").
digits	integer indicating the number of decimal places.

Details

These functions are methods for the generic function round. They apply numerical rounding to the spatial coordinates of the point pattern x.

Value

A point pattern object, of the same class as x.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rounding to determine whether numbers have been rounded.

round in the Base package.

Examples

```
round(cells, 1)
```

`rounding` *Detect Numerical Rounding*

Description

Given a numeric vector, or an object containing numeric spatial coordinates, determine whether the values have been rounded to a certain number of decimal places.

Usage

```
rounding(x)

## Default S3 method:
rounding(x)

## S3 method for class 'ppp'
rounding(x)

## S3 method for class 'pp3'
rounding(x)

## S3 method for class 'ppx'
rounding(x)
```

Arguments

`x`	A numeric vector, or an object containing numeric spatial coordinates.

Details

For a numeric vector `x`, this function determines whether the values have been rounded to a certain number of decimal places.

- If the entries of `x` are not all integers, then `rounding(x)` returns the smallest number of digits `d` after the decimal point such that `round(x, digits=d)` is identical to `x`. For example if `rounding(x) = 2` then the entries of `x` are rounded to 2 decimal places, and are multiples of 0.01.
- If all the entries of `x` are integers, then `rounding(x)` returns `-d`, where `d` is the smallest number of digits *before* the decimal point such that `round(x, digits=-d)` is identical to `x`. For example if `rounding(x) = -3` then the entries of `x` are multiples of 1000. If `rounding(x) = 0` then the entries of `x` are integers but not multiples of 10.
- If all entries of `x` are equal to 0, the rounding is not determined, and a value of `NULL` is returned.

For a point pattern (object of class `"ppp"`) or similar object `x` containing numeric spatial coordinates, this procedure is applied to the spatial coordinates.

Value

An integer.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

round.ppp

Examples

```
rounding(c(0.1, 0.3, 1.2))
rounding(c(1940, 1880, 2010))
rounding(0)
rounding(cells)
```

---

rPenttinen *Perfect Simulation of the Penttinen Process*

Description

Generate a random pattern of points, a simulated realisation of the Penttinen process, using a perfect simulation algorithm.

Usage

```
rPenttinen(beta, gamma=1, R, W = owin(), expand=TRUE, nsim=1, drop=TRUE)
```

Arguments

beta	intensity parameter (a positive number).
gamma	Interaction strength parameter (a number between 0 and 1).
R	disc radius (a non-negative number).
W	window (object of class "owin") in which to generate the random pattern.
expand	Logical. If FALSE, simulation is performed in the window W, which must be rectangular. If TRUE (the default), simulation is performed on a larger window, and the result is clipped to the original window W. Alternatively expand can be an object of class "rmhexpand" (see rmhexpand) determining the expansion method.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates a realisation of the Penttinen point process in the window `W` using a 'perfect simulation' algorithm.

Penttinen (1984, Example 2.1, page 18), citing Cormack (1979), described the pairwise interaction point process with interaction factor

$$h(d) = e^{\theta A(d)} = \gamma^{A(d)}$$

between each pair of points separated by a distance d. Here $A(d)$ is the area of intersection between two discs of radius R separated by a distance d, normalised so that $A(0) = 1$.

The simulation algorithm used to generate the point pattern is 'dominated coupling from the past' as implemented by Berthelsen and Møller (2002, 2003). This is a 'perfect simulation' or 'exact simulation' algorithm, so called because the output of the algorithm is guaranteed to have the correct probability distribution exactly (unlike the Metropolis-Hastings algorithm used in `rmh`, whose output is only approximately correct).

There is a tiny chance that the algorithm will run out of space before it has terminated. If this occurs, an error message will be generated.

Value

If `nsim = 1`, a point pattern (object of class `"ppp"`). If `nsim > 1`, a list of point patterns.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, based on original code for the Strauss process by Kasper Klitgaard Berthelsen.

References

Berthelsen, K.K. and Møller, J. (2002) A primer on perfect simulation for spatial point processes. *Bulletin of the Brazilian Mathematical Society* 33, 351-367.

Berthelsen, K.K. and Møller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* 30, 549-564.

Cormack, R.M. (1979) Spatial aspects of competition between individuals. Pages 151–212 in *Spatial and Temporal Analysis in Ecology*, eds. R.M. Cormack and J.K. Ord, International Co-operative Publishing House, Fairland, MD, USA.

Møller, J. and Waagepetersen, R. (2003). *Statistical Inference and Simulation for Spatial Point Processes.* Chapman and Hall/CRC.

Penttinen, A. (1984) *Modelling Interaction in Spatial Point Patterns: Parameter Estimation by the Maximum Likelihood Method.* Jyväskylä Studies in Computer Science, Economics and Statistics **7**, University of Jyväskylä, Finland.

See Also

`rmh`, `Penttinen`.

`rStrauss`, `rHardcore`, `rStraussHard`, `rDiggleGratton`, `rDGS`.

Examples

```
X <- rPenttinen(50, 0.5, 0.02)
```

`rpoint` *Generate N Random Points*

Description

Generate a random point pattern containing n independent, identically distributed random points with any specified distribution.

Usage

```
rpoint(n, f, fmax=NULL, win=unit.square(),
       ..., giveup=1000, verbose=FALSE,
       nsim=1, drop=TRUE)
```

Arguments

`n`	Number of points to generate.
`f`	The probability density of the points, possibly un-normalised. Either a constant, a function `f(x,y,...)`, or a pixel image object.
`fmax`	An upper bound on the values of `f`. If missing, this number will be estimated.
`win`	Window in which to simulate the pattern. Ignored if `f` is a pixel image.
`...`	Arguments passed to the function `f`.
`giveup`	Number of attempts in the rejection method after which the algorithm should stop trying to generate new points.
`verbose`	Flag indicating whether to report details of performance of the simulation algorithm.
`nsim`	Number of simulated realisations to be generated.
`drop`	Logical. If `nsim=1` and `drop=TRUE` (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates `n` independent, identically distributed random points with common probability density proportional to `f`.

The argument `f` may be

a numerical constant: uniformly distributed random points will be generated.

a function: random points will be generated in the window `win` with probability density proportional to `f(x,y,...)` where `x` and `y` are the cartesian coordinates. The function `f` must accept two *vectors* of coordinates `x,y` and return the corresponding vector of function values. Additional arguments `...` of any kind may be passed to the function.

a pixel image: if `f` is a pixel image object of class `"im"` (see `im.object`) then random points will be generated in the window of this pixel image, with probability density proportional to the pixel values of `f`.

The algorithm is as follows:

- If `f` is a constant, we invoke `runifpoint`.

- If f is a function, then we use the rejection method. Proposal points are generated from the uniform distribution. A proposal point (x, y) is accepted with probability `f(x,y,...)/fmax` and otherwise rejected. The algorithm continues until n points have been accepted. It gives up after `giveup * n` proposals if there are still fewer than n points.
- If f is a pixel image, then a random sequence of pixels is selected (using sample) with probabilities proportional to the pixel values of f. Then for each pixel in the sequence we generate a uniformly distributed random point in that pixel.

The algorithm for pixel images is more efficient than that for functions.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppp.object, owin.object, runifpoint

Examples

```
# 100 uniform random points in the unit square
X <- rpoint(100)

# 100 random points with probability density proportional to x^2 + y^2
X <- rpoint(100, function(x,y) { x^2 + y^2}, 1)

# `fmax' may be omitted
X <- rpoint(100, function(x,y) { x^2 + y^2})

# irregular window
data(letterR)
X <- rpoint(100, function(x,y) { x^2 + y^2}, win=letterR)

# make a pixel image
Z <- setcov(letterR)
# 100 points with density proportional to pixel values
X <- rpoint(100, Z)
```

---

rpoisline *Generate Poisson Random Line Process*

Description

Generate a random pattern of line segments obtained from the Poisson line process.

Usage

```
rpoisline(lambda, win=owin())
```

Arguments

lambda Intensity of the Poisson line process. A positive number.

win Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin.

Details

This algorithm generates a realisation of the uniform Poisson line process, and clips it to the window win.

The argument lambda must be a positive number. It controls the intensity of the process. The expected number of lines intersecting a convex region of the plane is equal to lambda times the perimeter length of the region. The expected total length of the lines crossing a region of the plane is equal to lambda * pi times the area of the region.

Value

A line segment pattern (an object of class "psp").

The result also has an attribute called "lines" (an object of class "infline" specifying the original infinite random lines) and an attribute "linemap" (an integer vector mapping the line segments to their parent lines).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

psp

Examples

```
# uniform Poisson line process with intensity 10,
# clipped to the unit square
rpoisline(10)
```

---

rpoislinetess *Poisson Line Tessellation*

Description

Generate a tessellation delineated by the lines of the Poisson line process

Usage

```
rpoislinetess(lambda, win = owin())
```

Arguments

lambda Intensity of the Poisson line process. A positive number.

win Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin. Currently, the window must be a rectangle.

Details

This algorithm generates a realisation of the uniform Poisson line process, and divides the window `win` into tiles separated by these lines.

The argument `lambda` must be a positive number. It controls the intensity of the process. The expected number of lines intersecting a convex region of the plane is equal to `lambda` times the perimeter length of the region. The expected total length of the lines crossing a region of the plane is equal to `lambda * pi` times the area of the region.

Value

A tessellation (object of class `"tess"`).

Also has an attribute `"lines"` containing the realisation of the Poisson line process, as an object of class `"infline"`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`rpoisline` to generate the lines only.

Examples

```
X <- rpoislinetess(3)
plot(as.im(X), main="rpoislinetess(3)")
plot(X, add=TRUE)
```

`rpoislpp` *Poisson Point Process on a Linear Network*

Description

Generates a realisation of the Poisson point process with specified intensity on the given linear network.

Usage

```
rpoislpp(lambda, L, ..., nsim=1, drop=TRUE)
```

Arguments

`lambda`	Intensity of the Poisson process. A single number, a `function(x,y)`, a pixel image (object of class `"im"`), or a vector of numbers, a list of functions, or a list of images.
`L`	A linear network (object of class `"linnet"`, see `linnet`). Can be omitted in some cases: see Details.
`...`	Arguments passed to `rpoisppOnLines`.
`nsim`	Number of simulated realisations to generate.

drop — Logical value indicating what to do when nsim=1. If drop=TRUE (the default), the result is a point pattern. If drop=FALSE, the result is a list with one entry which is a point pattern.

Details

This function uses rpoisppOnLines to generate the random points.

Argument L can be omitted, and defaults to as.linnet(lambda), when lambda is a function on a linear network (class "linfun") or a pixel image on a linear network ("linim").

Value

If nsim = 1 and drop=TRUE, a point pattern on the linear network, i.e.\ an object of class "lpp". Otherwise, a list of such point patterns.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

runiflpp, rlpp, lpp, linnet

Examples

```
X <- rpoislpp(5, simplenet)
plot(X)
# multitype
X <- rpoislpp(c(a=5, b=5), simplenet)
```

---

rpoispp — *Generate Poisson Point Pattern*

Description

Generate a random point pattern using the (homogeneous or inhomogeneous) Poisson process. Includes CSR (complete spatial randomness).

Usage

```
rpoispp(lambda, lmax=NULL, win=owin(), ...,
        nsim=1, drop=TRUE, ex=NULL, warnwin=TRUE)
```

Arguments

lambda	Intensity of the Poisson process. Either a single positive number, a function(x,y, ...), or a pixel image.
lmax	Optional. An upper bound for the value of lambda(x,y), if lambda is a function.
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin. Ignored if lambda is a pixel image.
...	Arguments passed to lambda if it is a function.

nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.
ex	Optional. A point pattern to use as the example. If ex is given and lambda,lmax,win are missing, then lambda and win will be calculated from the point pattern ex.
warnwin	Logical value specifying whether to issue a warning when win is ignored (which occurs when lambda is an image and win is present).

Details

If lambda is a single number, then this algorithm generates a realisation of the uniform Poisson process (also known as Complete Spatial Randomness, CSR) inside the window win with intensity lambda (points per unit area).

If lambda is a function, then this algorithm generates a realisation of the inhomogeneous Poisson process with intensity function lambda(x,y,...) at spatial location (x,y) inside the window win. The function lambda must work correctly with vectors x and y.

If lmax is given, it must be an upper bound on the values of lambda(x,y,...) for all locations (x, y) inside the window win. That is, we must have lambda(x,y,...) <= lmax for all locations (x,y). If this is not true then the results of the algorithm will be incorrect.

If lmax is missing or NULL, an approximate upper bound is computed by finding the maximum value of lambda(x,y,...) on a grid of locations (x,y) inside the window win, and adding a safety margin equal to 5 percent of the range of lambda values. This can be computationally intensive, so it is advisable to specify lmax if possible.

If lambda is a pixel image object of class "im" (see im.object), this algorithm generates a realisation of the inhomogeneous Poisson process with intensity equal to the pixel values of the image. (The value of the intensity function at an arbitrary location is the pixel value of the nearest pixel.) The argument win is ignored; the window of the pixel image is used instead. It will be converted to a rectangle if possible, using rescue.rectangle.

To generate an inhomogeneous Poisson process the algorithm uses "thinning": it first generates a uniform Poisson process of intensity lmax, then randomly deletes or retains each point, independently of other points, with retention probability $p(x,y) = \lambda(x,y)/\mathrm{lmax}$.

For *marked* point patterns, use rmpoispp.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Warning

Note that lambda is the **intensity**, that is, the expected number of points **per unit area**. The total number of points in the simulated pattern will be random with expected value mu = lambda * a where a is the area of the window win.

Reproducibility

The simulation algorithm, for the case where lambda is a pixel image, was changed in **spatstat** version 1.42-3. Set spatstat.options(fastpois=FALSE) to use the previous, slower algorithm, if it is desired to reproduce results obtained with earlier versions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

rmpoispp for Poisson *marked* point patterns, runifpoint for a fixed number of independent uniform random points; rpoint, rmpoint for a fixed number of independent random points with any distribution; rMaternI, rMaternII, rSSI, rStrauss, rstrat for random point processes with spatial inhibition or regularity; rThomas, rGaussPoisson, rMatClust, rcell for random point processes exhibiting clustering; rmh.default for Gibbs processes. See also ppp.object, owin.object.

Examples

```
# uniform Poisson process with intensity 100 in the unit square
pp <- rpoispp(100)

# uniform Poisson process with intensity 1 in a 10 x 10 square
pp <- rpoispp(1, win=owin(c(0,10),c(0,10)))
# plots should look similar !

# inhomogeneous Poisson process in unit square
# with intensity lambda(x,y) = 100 * exp(-3*x)
# Intensity is bounded by 100
pp <- rpoispp(function(x,y) {100 * exp(-3*x)}, 100)

# How to tune the coefficient of x
lamb <- function(x,y,a) { 100 * exp( - a * x)}
pp <- rpoispp(lamb, 100, a=3)

# pixel image
Z <- as.im(function(x,y){100 * sqrt(x+y)}, unit.square())
pp <- rpoispp(Z)

# randomising an existing point pattern
rpoispp(intensity(cells), win=Window(cells))
rpoispp(ex=cells)
```

---

rpoispp3 — *Generate Poisson Point Pattern in Three Dimensions*

Description

Generate a random three-dimensional point pattern using the homogeneous Poisson process.

Usage

```
rpoispp3(lambda, domain = box3(), nsim=1, drop=TRUE)
```

Arguments

lambda	Intensity of the Poisson process. A single positive number.
domain	Three-dimensional box in which the process should be generated. An object of class "box3".
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates a realisation of the homogeneous Poisson process in three dimensions, with intensity lambda (points per unit volume).

The realisation is generated inside the three-dimensional region domain which currently must be a rectangular box (object of class "box3").

Value

If nsim = 1 and drop=TRUE, a point pattern in three dimensions (an object of class "pp3"). If nsim > 1, a list of such point patterns.

Note

The intensity lambda is the expected number of points *per unit volume*.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

runifpoint3, pp3, box3

Examples

```
X <- rpoispp3(50)
```

---

rpoisppOnLines *Generate Poisson Point Pattern on Line Segments*

Description

Given a line segment pattern, generate a Poisson random point pattern on the line segments.

Usage

```
rpoisppOnLines(lambda, L, lmax = NULL, ..., nsim=1)
```

Arguments

lambda	Intensity of the Poisson process. A single number, a function(x,y), a pixel image (object of class "im"), or a vector of numbers, a list of functions, or a list of images.
L	Line segment pattern (object of class "psp") on which the points should be generated.
lmax	Optional upper bound (for increased computational efficiency). A known upper bound for the values of lambda, if lambda is a function or a pixel image. That is, lmax should be a number which is known to be greater than or equal to all values of lambda.
...	Additional arguments passed to lambda if it is a function.
nsim	Number of simulated realisations to be generated.

Details

This command generates a Poisson point process on the one-dimensional system of line segments in L. The result is a point pattern consisting of points lying on the line segments in L. The number of random points falling on any given line segment follows a Poisson distribution. The patterns of points on different segments are independent.

The intensity lambda is the expected number of points per unit **length** of line segment. It may be constant, or it may depend on spatial location.

In order to generate an unmarked Poisson process, the argument lambda may be a single number, or a function(x,y), or a pixel image (object of class "im").

In order to generate a *marked* Poisson process, lambda may be a numeric vector, a list of functions, or a list of images, each entry giving the intensity for a different mark value.

If lambda is not numeric, then the (Lewis-Shedler) rejection method is used. The rejection method requires knowledge of lmax, the maximum possible value of lambda. This should be either a single number, or a numeric vector of the same length as lambda. If lmax is not given, it will be computed approximately, by sampling many values of lambda.

If lmax is given, then it **must** be larger than any possible value of lambda, otherwise the results of the algorithm will be incorrect.

Value

If nsim = 1, a point pattern (object of class "ppp") in the same window as L. If nsim > 1, a list of such point patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

psp, ppp, runifpointOnLines, rpoispp

Examples

```
live <- interactive()
L <- psp(runif(10), runif(10), runif(10), runif(10),  window=owin())
if(live) plot(L, main="")

# uniform intensity
Y <- rpoisppOnLines(4, L)
if(live) plot(Y, add=TRUE, pch="+")

# uniform MARKED process with types 'a' and 'b'
Y <- rpoisppOnLines(c(a=4, b=5), L)
if(live) {
  plot(L, main="")
  plot(Y, add=TRUE, pch="+")
}

# intensity is a function
Y <- rpoisppOnLines(function(x,y){ 10 * x^2}, L, 10)
if(live) {
  plot(L, main="")
  plot(Y, add=TRUE, pch="+")
}

# intensity is an image
Z <- as.im(function(x,y){10 * sqrt(x+y)}, unit.square())
Y <- rpoisppOnLines(Z, L, 15)
if(live) {
 plot(L, main="")
 plot(Y, add=TRUE, pch="+")
}
```

---

rpoisppx *Generate Poisson Point Pattern in Any Dimensions*

Description

Generate a random multi-dimensional point pattern using the homogeneous Poisson process.

Usage

```
rpoisppx(lambda, domain, nsim=1, drop=TRUE)
```

Arguments

lambda	Intensity of the Poisson process. A single positive number.
domain	Multi-dimensional box in which the process should be generated. An object of class "boxx".
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a single point pattern.

Details

This function generates a realisation of the homogeneous Poisson process in multi dimensions, with intensity `lambda` (points per unit volume).

The realisation is generated inside the multi-dimensional region `domain` which currently must be a rectangular box (object of class "boxx").

Value

If `nsim = 1` and `drop=TRUE`, a point pattern (an object of class "ppx"). If `nsim > 1` or `drop=FALSE`, a list of such point patterns.

Note

The intensity `lambda` is the expected number of points *per unit volume*.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

runifpointx, ppx, boxx

Examples

```
w <- boxx(x=c(0,1), y=c(0,1), z=c(0,1), t=c(0,3))
X <- rpoisppx(10, w)
```

---

rPoissonCluster *Simulate Poisson Cluster Process*

Description

Generate a random point pattern, a realisation of the general Poisson cluster process.

Usage

```
rPoissonCluster(kappa, expand, rcluster, win = owin(c(0,1),c(0,1)),
                ..., lmax=NULL, nsim=1, drop=TRUE, saveparents=TRUE)
```

Arguments

kappa	Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.
expand	Size of the expansion of the simulation window for generating parent points. A single non-negative number.
rcluster	A function which generates random clusters.
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin.

...	Arguments passed to rcluster
lmax	Optional. Upper bound on the values of kappa when kappa is a function or pixel image.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.
saveparents	Logical value indicating whether to save the locations of the parent points as an attribute.

Details

This algorithm generates a realisation of the general Poisson cluster process, with the cluster mechanism given by the function rcluster.

First, the algorithm generates a Poisson point process of "parent" points with intensity kappa in an expanded window as explained below.. Here kappa may be a single positive number, a function kappa(x, y), or a pixel image object of class "im" (see im.object). See rpoispp for details.

Second, each parent point is replaced by a random cluster of points, created by calling the function rcluster. These clusters are combined together to yield a single point pattern, and the restriction of this pattern to the window win is then returned as the result of rPoissonCluster.

The expanded window consists of as.rectangle(win) extended by the amount expand in each direction. The size of the expansion is saved in the attribute "expand" and may be extracted by attr(X, "expand") where X is the generated point pattern.

The function rcluster should expect to be called as rcluster(xp[i],yp[i],...) for each parent point at a location (xp[i],yp[i]). The return value of rcluster should be a list with elements x,y which are vectors of equal length giving the absolute x and y coordinates of the points in the cluster.

If the return value of rcluster is a point pattern (object of class "ppp") then it may have marks. The result of rPoissonCluster will then be a marked point pattern.

If required, the intermediate stages of the simulation (the parents and the individual clusters) can also be extracted from the return value of rPoissonCluster through the attributes "parents" and "parentid". The attribute "parents" is the point pattern of parent points. The attribute "parentid" is an integer vector specifying the parent for each of the points in the simulated pattern. (If these data are not required, it is more efficient to set saveparents=FALSE.)

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Additionally, some intermediate results of the simulation are returned as attributes of the point pattern: see Details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rpoispp, rMatClust, rThomas, rCauchy, rVarGamma, rNeymanScott, rGaussPoisson.

Examples

```
# each cluster consist of 10 points in a disc of radius 0.2
nclust <- function(x0, y0, radius, n) {
            return(runifdisc(n, radius, centre=c(x0, y0)))
          }
plot(rPoissonCluster(10, 0.2, nclust, radius=0.2, n=5))

# multitype Neyman-Scott process (each cluster is a multitype process)
nclust2 <- function(x0, y0, radius, n, types=c("a", "b")) {
   X <- runifdisc(n, radius, centre=c(x0, y0))
   M <- sample(types, n, replace=TRUE)
   marks(X) <- M
   return(X)
}
plot(rPoissonCluster(15,0.1,nclust2, radius=0.1, n=5))
```

---

rppm *Recursively Partitioned Point Process Model*

Description

Fits a recursive partition model to point pattern data.

Usage

```
rppm(..., rpargs=list())
```

Arguments

...	Arguments passed to ppm specifying the point pattern data and the explanatory covariates.
rpargs	Optional list of arguments passed to rpart controlling the recursive partitioning procedure.

Details

This function attempts to find a simple rule for predicting low and high intensity regions of points in a point pattern, using explanatory covariates.

The arguments ... specify the point pattern data and explanatory covariates in the same way as they would be in the function ppm.

The recursive partitioning algorithm rpart is then used to find a partitioning rule.

Value

An object of class "rppm". There are methods for print, plot, fitted, predict and prune for this class.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984) *Classification and Regression Trees*. Wadsworth.

See Also

plot.rppm, predict.rppm, prune.rppm.

Examples

```
# New Zealand trees data: trees planted along border
# Use covariates 'x', 'y'
nzfit <- rppm(nztrees ~ x + y)
nzfit
prune(nzfit, cp=0.035)
# Murchison gold data: numeric and logical covariates
mur <- solapply(murchison, rescale, s=1000, unitname="km")
mur$dfault <- distfun(mur$faults)
#
mfit <- rppm(gold ~ dfault + greenstone, data=mur)
mfit
# Gorillas data: factor covariates
#          (symbol '.' indicates 'all variables')
gfit <- rppm(unmark(gorillas) ~ . , data=gorillas.extra)
gfit
```

---

rQuasi *Generate Quasirandom Point Pattern in Given Window*

Description

Generates a quasirandom pattern of points in any two-dimensional window.

Usage

```
rQuasi(n, W, type = c("Halton", "Hammersley"), ...)
```

Arguments

n	Maximum number of points to be generated.
W	Window (object of class "owin") in which to generate the points.
type	String identifying the quasirandom generator.
...	Arguments passed to the quasirandom generator.

Details

This function generates a quasirandom point pattern, using the quasirandom sequence generator Halton or Hammersley as specified.

If W is a rectangle, exactly n points will be generated.

If W is not a rectangle, n points will be generated in the containing rectangle as.rectangle(W), and only the points lying inside W will be retained.

Value

Point pattern (object of class "ppp") inside the window W.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

See Also

Halton

Examples

```
plot(rQuasi(256, letterR))
```

---

rshift *Random Shift*

Description

Randomly shifts the points of a point pattern or line segment pattern. Generic.

Usage

```
rshift(X, ...)
```

Arguments

X	Pattern to be subjected to a random shift. A point pattern (class "ppp"), a line segment pattern (class "psp") or an object of class "splitppp".
...	Arguments controlling the generation of the random shift vector, or specifying which parts of the pattern will be shifted.

Details

This operation applies a random shift (vector displacement) to the points in a point pattern, or to the segments in a line segment pattern.

The argument X may be

- a point pattern (an object of class "ppp")
- a line segment pattern (an object of class "psp")
- an object of class "splitppp" (basically a list of point patterns, obtained from split.ppp).

The function rshift is generic, with methods for the three classes "ppp", "psp" and "splitppp".

See the help pages for these methods, rshift.ppp, rshift.psp and rshift.splitppp, for further information.

Value

An object of the same type as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rshift.ppp, rshift.psp, rshift.splitppp

---

rshift.ppp *Randomly Shift a Point Pattern*

Description

Randomly shifts the points of a point pattern.

Usage

```
   ## S3 method for class 'ppp'
rshift(X, ..., which=NULL, group)
```

Arguments

X	Point pattern to be subjected to a random shift. An object of class "ppp"
...	Arguments that determine the random shift. See Details.
group	Optional. Factor specifying a grouping of the points of X, or NULL indicating that all points belong to the same group. Each group will be shifted together, and separately from other groups. By default, points in a marked point pattern are grouped according to their mark values, while points in an unmarked point pattern are treated as a single group.
which	Optional. Identifies which groups of the pattern will be shifted, while other groups are not shifted. A vector of levels of group.

Details

This operation randomly shifts the locations of the points in a point pattern.

The function rshift is generic. This function rshift.ppp is the method for point patterns.

The most common use of this function is to shift the points in a multitype point pattern. By default, points of the same type are shifted in parallel (i.e. points of a common type are shifted by a common displacement vector), and independently of other types. This is useful for testing the hypothesis of independence of types (the null hypothesis that the sub-patterns of points of each type are independent point processes).

In general the points of X are divided into groups, then the points within a group are shifted by a common random displacement vector. Different groups of points are shifted independently. The grouping is determined as follows:

- If the argument group is present, then this determines the grouping.
- Otherwise, if X is a multitype point pattern, the marks determine the grouping.
- Otherwise, all points belong to a single group.

The argument group should be a factor, of length equal to the number of points in X. Alternatively group may be NULL, which specifies that all points of X belong to a single group.

By default, every group of points will be shifted. The argument which indicates that only some of the groups should be shifted, while other groups should be left unchanged. which must be a vector of levels of group (for example, a vector of types in a multitype pattern) indicating which groups are to be shifted.

The displacement vector, i.e. the vector by which the data points are shifted, is generated at random. Parameters that control the randomisation and the handling of edge effects are passed through the ... argument. They are

radius,width,height Parameters of the random shift vector.

edge String indicating how to deal with edges of the pattern. Options are "torus", "erode" and "none".

clip Optional. Window to which the final point pattern should be clipped.

If the window is a rectangle, the *default* behaviour is to generate a displacement vector at random with equal probability for all possible displacements. This means that the x and y coordinates of the displacement vector are independent random variables, uniformly distributed over the range of possible coordinates.

Alternatively, the displacement vector can be generated by another random mechanism, controlled by the arguments radius, width and height.

rectangular: if width and height are given, then the displacement vector is uniformly distributed in a rectangle of these dimensions, centred at the origin. The maximum possible displacement in the x direction is width/2. The maximum possible displacement in the y direction is height/2. The x and y displacements are independent. (If width and height are actually equal to the dimensions of the observation window, then this is equivalent to the default.)

radial: if radius is given, then the displacement vector is generated by choosing a random point inside a disc of the given radius, centred at the origin, with uniform probability density over the disc. Thus the argument radius determines the maximum possible displacement distance. The argument radius is incompatible with the arguments width and height.

The argument edge controls what happens when a shifted point lies outside the window of X. Options are:

"none": Points shifted outside the window of X simply disappear.

"torus": Toroidal or periodic boundary. Treat opposite edges of the window as identical, so that a point which disappears off the right-hand edge will re-appear at the left-hand edge. This is called a "toroidal shift" because it makes the rectangle topologically equivalent to the surface of a torus (doughnut).

The window must be a rectangle. Toroidal shifts are undefined if the window is non-rectangular.

"erode": Clip the point pattern to a smaller window.

If the random displacements are generated by a radial mechanism (see above), then the window of X is eroded by a distance equal to the value of the argument radius, using erosion.

If the random displacements are generated by a rectangular mechanism, then the window of X is (if it is not rectangular) eroded by a distance max(height,width) using erosion; or (if it is rectangular) trimmed by a margin of width `width` at the left and right sides and trimmed by a margin of height `height` at the top and bottom.

The rationale for this is that the clipping window is the largest window for which edge effects can be ignored.

The optional argument `clip` specifies a smaller window to which the pattern should be restricted.

Value

A point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rshift, rshift.psp

Examples

```
data(amacrine)

# random toroidal shift
# shift "on" and "off" points separately
X <- rshift(amacrine)

# shift "on" points and leave "off" points fixed
X <- rshift(amacrine, which="on")

# shift all points simultaneously
X <- rshift(amacrine, group=NULL)

# maximum displacement distance 0.1 units
X <- rshift(amacrine, radius=0.1)

# shift with erosion
X <- rshift(amacrine, radius=0.1, edge="erode")
```

---

rshift.psp *Randomly Shift a Line Segment Pattern*

Description

Randomly shifts the segments in a line segment pattern.

Usage

```
## S3 method for class 'psp'
rshift(X, ..., group=NULL, which=NULL)
```

Arguments

X	Line segment pattern to be subjected to a random shift. An object of class "psp".
...	Arguments controlling the randomisation and the handling of edge effects. See rshift.ppp.
group	Optional. Factor specifying a grouping of the line segments of X, or NULL indicating that all line segments belong to the same group. Each group will be shifted together, and separately from other groups.
which	Optional. Identifies which groups of the pattern will be shifted, while other groups are not shifted. A vector of levels of group.

Details

This operation randomly shifts the locations of the line segments in a line segment pattern.

The function rshift is generic. This function rshift.psp is the method for line segment patterns.

The line segments of X are first divided into groups, then the line segments within a group are shifted by a common random displacement vector. Different groups of line segments are shifted independently. If the argument group is present, then this determines the grouping. Otherwise, all line segments belong to a single group.

The argument group should be a factor, of length equal to the number of line segments in X. Alternatively group may be NULL, which specifies that all line segments of X belong to a single group.

By default, every group of line segments will be shifted. The argument which indicates that only some of the groups should be shifted, while other groups should be left unchanged. which must be a vector of levels of group indicating which groups are to be shifted.

The displacement vector, i.e. the vector by which the data line segments are shifted, is generated at random. The *default* behaviour is to generate a displacement vector at random with equal probability for all possible displacements. This means that the x and y coordinates of the displacement vector are independent random variables, uniformly distributed over the range of possible coordinates.

Alternatively, the displacement vector can be generated by another random mechanism, controlled by the arguments radius, width and height.

rectangular: if width and height are given, then the displacement vector is uniformly distributed in a rectangle of these dimensions, centred at the origin. The maximum possible displacement in the x direction is width/2. The maximum possible displacement in the y direction is height/2. The x and y displacements are independent. (If width and height are actually equal to the dimensions of the observation window, then this is equivalent to the default.)

radial: if radius is given, then the displacement vector is generated by choosing a random line segment inside a disc of the given radius, centred at the origin, with uniform probability density over the disc. Thus the argument radius determines the maximum possible displacement distance. The argument radius is incompatible with the arguments width and height.

The argument edge controls what happens when a shifted line segment lies partially or completely outside the window of X. Currently the only option is "erode" which specifies that the segments will be clipped to a smaller window.

The optional argument clip specifies a smaller window to which the pattern should be restricted.

Value

A line segment pattern (object of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rshift, rshift.ppp

Examples

```
X <- psp(runif(20), runif(20), runif(20), runif(20),  window=owin())
Y <- rshift(X, radius=0.1)
```

---

rshift.splitppp *Randomly Shift a List of Point Patterns*

Description

Randomly shifts each point pattern in a list of point patterns.

Usage

```
    ## S3 method for class 'splitppp'
rshift(X, ..., which=seq_along(X))
```

Arguments

X	An object of class "splitppp". Basically a list of point patterns.
...	Parameters controlling the generation of the random shift vector and the handling of edge effects. See rshift.ppp.
which	Optional. Identifies which patterns will be shifted, while other patterns are not shifted. Any valid subset index for X.

Details

This operation applies a random shift to each of the point patterns in the list X.

The function rshift is generic. This function rshift.splitppp is the method for objects of class "splitppp", which are essentially lists of point patterns, created by the function split.ppp.

By default, every pattern in the list X will be shifted. The argument which indicates that only some of the patterns should be shifted, while other groups should be left unchanged. which can be any valid subset index for X.

Each point pattern in the list X (or each pattern in X[which]) is shifted by a random displacement vector. The shifting is performed by rshift.ppp.

See the help page for rshift.ppp for details of the other arguments.

Value

Another object of class "splitppp".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rshift, rshift.ppp

Examples

```
data(amacrine)
Y <- split(amacrine)

# random toroidal shift
# shift "on" and "off" points separately
X <- rshift(Y)

# shift "on" points and leave "off" points fixed
X <- rshift(Y, which="on")

# maximum displacement distance 0.1 units
X <- rshift(Y, radius=0.1)

# shift with erosion
X <- rshift(Y, radius=0.1, edge="erode")
```

---

rSSI *Simulate Simple Sequential Inhibition*

Description

Generate a random point pattern, a realisation of the Simple Sequential Inhibition (SSI) process.

Usage

```
rSSI(r, n=Inf, win = square(1), giveup = 1000, x.init=NULL, ...,
     f=NULL, fmax=NULL, nsim=1, drop=TRUE)
```

Arguments

r	Inhibition distance.
n	Maximum number of points allowed. If n is finite, stop when the *total* number of points in the point pattern reaches n. If n is infinite (the default), stop only when it is apparently impossible to add any more points. See **Details**.
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin. The default window is the unit square, unless x.init is specified, when the default window is the window of x.init.
giveup	Number of rejected proposals after which the algorithm should terminate.
x.init	Optional. Initial configuration of points. A point pattern (object of class "ppp"). The pattern returned by rSSI consists of this pattern together with the points added via simple sequential inhibition. See **Details**.

...	Ignored.
f,fmax	Optional arguments passed to rpoint to specify a non-uniform probability density for the random points.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This algorithm generates one or more realisations of the Simple Sequential Inhibition point process inside the window win.

Starting with an empty window (or with the point pattern x.init if specified), the algorithm adds points one-by-one. Each new point is generated uniformly in the window and independently of preceding points. If the new point lies closer than r units from an existing point, then it is rejected and another random point is generated. The algorithm terminates when either

(a) the desired number n of points is reached, or

(b) the current point configuration has not changed for giveup iterations, suggesting that it is no longer possible to add new points.

If n is infinite (the default) then the algorithm terminates only when (b) occurs. The result is sometimes called a *Random Sequential Packing*.

Note that argument n specifies the maximum permitted **total** number of points in the pattern returned by rSSI(). If x.init is not NULL then the number of points that are *added* is at most n - npoints(x.init) if n is finite.

Thus if x.init is not NULL then argument n must be at least as large as npoints(x.init), otherwise an error is given. If n==npoints(x.init) then a warning is given and the call to rSSI() has no real effect; x.init is returned.

There is no requirement that the points of x.init be at a distance at least r from each other. All of the *added* points will be at a distance at least r from each other and from any point of x.init.

The points will be generated inside the window win and the result will be a point pattern in the same window.

The default window is the unit square, win = square(1), unless x.init is specified, when the default is win=Window(x.init), the window of x.init.

If both win and x.init are specified, and if the two windows are different, then a warning will be issued. Any points of x.init lying outside win will be removed, with a warning.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

rpoispp, rMaternI, rMaternII.

Examples

```
Vinf <- rSSI(0.07)

V100 <- rSSI(0.07, 100)

X <- runifpoint(100)
Y <- rSSI(0.03,142,x.init=X) # Y consists of X together with
                             # 42 added points.
plot(Y, main="rSSI")
plot(X,add=TRUE,chars=20,cols="red")

## inhomogeneous
Z <- rSSI(0.07, 50, f=function(x,y){x})
plot(Z)
```

---

rstrat *Simulate Stratified Random Point Pattern*

Description

Generates a "stratified random" pattern of points in a window, by dividing the window into rectangular tiles and placing k random points independently in each tile.

Usage

```
rstrat(win=square(1), nx, ny=nx, k = 1, nsim=1, drop=TRUE)
```

Arguments

win	A window. An object of class owin, or data in any format acceptable to as.owin().
nx	Number of tiles in each column.
ny	Number of tiles in each row.
k	Number of random points to generate in each tile.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates a random pattern of points in a "stratified random" sampling design. It can be useful for generating random spatial sampling points.

The bounding rectangle of win is divided into a regular $nx \times ny$ grid of rectangular tiles. In each tile, k random points are generated independently with a uniform distribution in that tile.

Some of these grid points may lie outside the window win: if they do, they are deleted.

The result is a point pattern inside the window win.

This function is useful in creating dummy points for quadrature schemes (see quadscheme) as well as in simulating random point patterns.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rsyst, runifpoint, quadscheme

Examples

```
X <- rstrat(nx=10)
plot(X)

# polygonal boundary
data(letterR)
X <- rstrat(letterR, 5, 10, k=3)
plot(X)
```

---

rStrauss *Perfect Simulation of the Strauss Process*

Description

Generate a random pattern of points, a simulated realisation of the Strauss process, using a perfect simulation algorithm.

Usage

```
rStrauss(beta, gamma = 1, R = 0, W = owin(), expand=TRUE, nsim=1, drop=TRUE)
```

Arguments

beta	intensity parameter (a positive number).
gamma	interaction parameter (a number between 0 and 1, inclusive).
R	interaction radius (a non-negative number).
W	window (object of class "owin") in which to generate the random pattern.
expand	Logical. If FALSE, simulation is performed in the window W, which must be rectangular. If TRUE (the default), simulation is performed on a larger window, and the result is clipped to the original window W. Alternatively expand can be an object of class "rmhexpand" (see rmhexpand) determining the expansion method.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates a realisation of the Strauss point process in the window `W` using a 'perfect simulation' algorithm.

The Strauss process (Strauss, 1975; Kelly and Ripley, 1976) is a model for spatial inhibition, ranging from a strong 'hard core' inhibition to a completely random pattern according to the value of `gamma`.

The Strauss process with interaction radius R and parameters β and γ is the pairwise interaction point process with probability density

$$f(x_1, \ldots, x_n) = \alpha \beta^{n(x)} \gamma^{s(x)}$$

where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, $s(x)$ is the number of distinct unordered pairs of points that are closer than R units apart, and α is the normalising constant. Intuitively, each point of the pattern contributes a factor β to the probability density, and each pair of points closer than r units apart contributes a factor γ to the density.

The interaction parameter γ must be less than or equal to 1 in order that the process be well-defined (Kelly and Ripley, 1976). This model describes an "ordered" or "inhibitive" pattern. If $\gamma = 1$ it reduces to a Poisson process (complete spatial randomness) with intensity β. If $\gamma = 0$ it is called a "hard core process" with hard core radius $R/2$, since no pair of points is permitted to lie closer than R units apart.

The simulation algorithm used to generate the point pattern is 'dominated coupling from the past' as implemented by Berthelsen and Møller (2002, 2003). This is a 'perfect simulation' or 'exact simulation' algorithm, so called because the output of the algorithm is guaranteed to have the correct probability distribution exactly (unlike the Metropolis-Hastings algorithm used in `rmh`, whose output is only approximately correct).

There is a tiny chance that the algorithm will run out of space before it has terminated. If this occurs, an error message will be generated.

Value

If `nsim = 1`, a point pattern (object of class `"ppp"`). If `nsim > 1`, a list of point patterns.

Author(s)

Kasper Klitgaard Berthelsen, adapted for **spatstat** by Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Berthelsen, K.K. and Møller, J. (2002) A primer on perfect simulation for spatial point processes. *Bulletin of the Brazilian Mathematical Society* 33, 351-367.

Berthelsen, K.K. and Møller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* 30, 549-564.

Kelly, F.P. and Ripley, B.D. (1976) On Strauss's model for clustering. *Biometrika* **63**, 357–360.

Møller, J. and Waagepetersen, R. (2003). *Statistical Inference and Simulation for Spatial Point Processes.* Chapman and Hall/CRC.

Strauss, D.J. (1975) A model for clustering. *Biometrika* **62**, 467–475.

See Also

`rmh`, `Strauss`, `rHardcore`, `rStraussHard`, `rDiggleGratton`, `rDGS`, `rPenttinen`.

Examples

```
X <- rStrauss(0.05,0.2,1.5,square(141.4))
Z <- rStrauss(100,0.7,0.05)
```

`rStraussHard` *Perfect Simulation of the Strauss-Hardcore Process*

Description

Generate a random pattern of points, a simulated realisation of the Strauss-Hardcore process, using a perfect simulation algorithm.

Usage

```
rStraussHard(beta, gamma = 1, R = 0, H = 0, W = owin(),
              expand=TRUE, nsim=1, drop=TRUE)
```

Arguments

`beta`	intensity parameter (a positive number).
`gamma`	interaction parameter (a number between 0 and 1, inclusive).
`R`	interaction radius (a non-negative number).
`H`	hard core distance (a non-negative number smaller than `R`).
`W`	window (object of class `"owin"`) in which to generate the random pattern. Currently this must be a rectangular window.
`expand`	Logical. If `FALSE`, simulation is performed in the window `W`, which must be rectangular. If `TRUE` (the default), simulation is performed on a larger window, and the result is clipped to the original window `W`. Alternatively `expand` can be an object of class `"rmhexpand"` (see `rmhexpand`) determining the expansion method.
`nsim`	Number of simulated realisations to be generated.
`drop`	Logical. If `nsim=1` and `drop=TRUE` (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates a realisation of the Strauss-Hardcore point process in the window `W` using a 'perfect simulation' algorithm.

The Strauss-Hardcore process is described in `StraussHard`.

The simulation algorithm used to generate the point pattern is 'dominated coupling from the past' as implemented by Berthelsen and Møller (2002, 2003). This is a 'perfect simulation' or 'exact simulation' algorithm, so called because the output of the algorithm is guaranteed to have the correct probability distribution exactly (unlike the Metropolis-Hastings algorithm used in `rmh`, whose output is only approximately correct).

A limitation of the perfect simulation algorithm is that the interaction parameter γ must be less than or equal to 1. To simulate a Strauss-hardcore process with $\gamma > 1$, use `rmh`.

There is a tiny chance that the algorithm will run out of space before it has terminated. If this occurs, an error message will be generated.

Value

If `nsim = 1`, a point pattern (object of class "ppp"). If `nsim > 1`, a list of point patterns.

Author(s)

Kasper Klitgaard Berthelsen and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Berthelsen, K.K. and Møller, J. (2002) A primer on perfect simulation for spatial point processes. *Bulletin of the Brazilian Mathematical Society* 33, 351-367.

Berthelsen, K.K. and Møller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* 30, 549-564.

Møller, J. and Waagepetersen, R. (2003). *Statistical Inference and Simulation for Spatial Point Processes.* Chapman and Hall/CRC.

See Also

`rmh`, `StraussHard`.

`rHardcore`, `rStrauss`, `rDiggleGratton`, `rDGS`, `rPenttinen`.

Examples

```
Z <- rStraussHard(100,0.7,0.05,0.02)
```

`rsyst` *Simulate systematic random point pattern*

Description

Generates a "systematic random" pattern of points in a window, consisting of a grid of equally-spaced points with a random common displacement.

Usage

```
rsyst(win=square(1), nx=NULL, ny=nx, ..., dx=NULL, dy=dx,
      nsim=1, drop=TRUE)
```

Arguments

`win`	A window. An object of class `owin`, or data in any format acceptable to `as.owin()`.
`nx`	Number of columns of grid points in the window. Incompatible with `dx`.
`ny`	Number of rows of grid points in the window. Incompatible with `dy`.
`...`	Ignored.
`dx`	Spacing of grid points in x direction. Incompatible with `nx`.
`dy`	Spacing of grid points in y direction. Incompatible with `ny`.
`nsim`	Number of simulated realisations to be generated.
`drop`	Logical. If `nsim=1` and `drop=TRUE` (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates a "systematic random" pattern of points in the window `win`. The pattern consists of a rectangular grid of points with a random common displacement.

The grid spacing in the x direction is determined either by the number of columns `nx` or by the horizontal spacing `dx`. The grid spacing in the y direction is determined either by the number of rows `ny` or by the vertical spacing `dy`.

The grid is then given a random displacement (the common displacement of the grid points is a uniformly distributed random vector in the tile of dimensions `dx, dy`).

Some of the resulting grid points may lie outside the window `win`: if they do, they are deleted. The result is a point pattern inside the window `win`.

This function is useful in creating dummy points for quadrature schemes (see `quadscheme`) as well as in simulating random point patterns.

Value

A point pattern (an object of class `"ppp"`) if `nsim=1`, or a list of point patterns if `nsim > 1`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`rstrat`, `runifpoint`, `quadscheme`

Examples

```
X <- rsyst(nx=10)
plot(X)

# polygonal boundary
data(letterR)
X <- rsyst(letterR, 5, 10)
plot(X)
```

`rtemper` *Simulated Annealing or Simulated Tempering for Gibbs Point Processes*

Description

Performs simulated annealing or simulated tempering for a Gibbs point process model using a specified annealing schedule.

Usage

```
rtemper(model, invtemp, nrep, ..., start = NULL, verbose = FALSE)
```

Arguments

model	A Gibbs point process model: a fitted Gibbs point process model (object of class "ppm"), or any data acceptable to rmhmodel.
invtemp	A numeric vector of positive numbers. The sequence of values of inverse temperature that will be used.
nrep	An integer vector of the same length as invtemp. The value nrep[i] specifies the number of steps of the Metropolis-Hastings algorithm that will be performed at inverse temperature invtemp[i].
start	Initial starting state for the simulation. Any data acceptable to rmhstart.
...	Additional arguments passed to rmh.default.
verbose	Logical value indicating whether to print progress reports.

Details

The Metropolis-Hastings simulation algorithm rmh is run for nrep[1] steps at inverse temperature invtemp[1], then for nrep[2] steps at inverse temperature invtemp[2], and so on.

Setting the inverse temperature to a value α means that the probability density of the Gibbs model, $f(x)$, is replaced by $g(x) = C\,f(x)^\alpha$ where C is a normalising constant depending on α. Larger values of α exaggerate the high and low values of probability density, while smaller values of α flatten out the probability density.

For example if the original model is a Strauss process, the modified model is close to a hard core process for large values of inverse temperature, and close to a Poisson process for small values of inverse temperature.

Value

A point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

rmh.default, rmh.

Examples

```
stra <- rmhmodel(cif="strauss",
                 par=list(beta=2,gamma=0.2,r=0.7),
                 w=square(10))
nr <- if(interactive()) 1e5 else 1e4
Y <- rtemper(stra, c(1, 2, 4, 8), nr * (1:4), verbose=TRUE)
```

rthin *Random Thinning*

Description

Applies independent random thinning to a point pattern.

Usage

```
rthin(X, P, ..., nsim=1, drop=TRUE)
```

Arguments

X	A point pattern (object of class "ppp" or "lpp") that will be thinned.
P	Data giving the retention probabilities, i.e. the probability that each point in X will be retained. Either a single number, or a vector of numbers, or a function(x,y) in the R language, or a function object (class "funxy" or "linfun"), or a pixel image (object of class "im" or "linim").
...	Additional arguments passed to P, if it is a function.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

In a random thinning operation, each point of the pattern X is randomly either deleted or retained (i.e. not deleted). The result is a point pattern, consisting of those points of X that were retained.

Independent random thinning means that the retention/deletion of each point is independent of other points.

The argument P determines the probability of **retaining** each point. It may be

a single number, so that each point will be retained with the same probability P;

a vector of numbers, so that the ith point of X will be retained with probability P[i];

a function P(x,y)**,** so that a point at a location (x,y) will be retained with probability P(x,y);

an object of class "funxy" **or** "linfun"**,** so that points in the pattern X will be retained with probabilities P(X);

a pixel image, containing values of the retention probability for all locations in a region encompassing the point pattern.

If P is a function P(x,y), it should be 'vectorised', that is, it should accept vector arguments x,y and should yield a numeric vector of the same length. The function may have extra arguments which are passed through the ... argument.

Value

A point pattern (object of class "ppp" or "lpp") if nsim=1, or a list of point patterns if nsim > 1.

Reproducibility

The algorithm for random thinning was changed in **spatstat** version `1.42-3`. Set `spatstat.options(fastthin=FALSE)` to use the previous, slower algorithm, if it is desired to reproduce results obtained with earlier versions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

Examples

```
plot(redwood, main="thinning")

# delete 20% of points
Y <- rthin(redwood, 0.8)
points(Y, col="green", cex=1.4)

# function
f <- function(x,y) { ifelse(x < 0.4, 1, 0.5) }
Y <- rthin(redwood, f)

# pixel image
Z <- as.im(f, Window(redwood))
Y <- rthin(redwood, Z)

# pattern on a linear network
A <- runiflpp(30, simplenet)
B <- rthin(A, 0.2)
g <- function(x,y,seg,tp) { ifelse(y < 0.4, 1, 0.5) }
B <- rthin(A, linfun(g, simplenet))
```

`rThomas` *Simulate Thomas Process*

Description

Generate a random point pattern, a realisation of the Thomas cluster process.

Usage

```
rThomas(kappa, scale, mu, win = owin(c(0,1),c(0,1)),
        nsim=1, drop=TRUE,
        saveLambda=FALSE, expand = 4*scale, ...,
        poisthresh=1e-6, saveparents=TRUE)
```

Arguments

`kappa`	Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.
`scale`	Standard deviation of random displacement (along each coordinate axis) of a point from its cluster centre.

mu	Mean number of points per cluster (a single positive number) or reference intensity for the cluster points (a function or a pixel image).
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.
saveLambda	Logical. If TRUE then the random intensity corresponding to the simulated parent points will also be calculated and saved, and returns as an attribute of the point pattern.
expand	Numeric. Size of window expansion for generation of parent points. Has a sensible default.
...	Passed to clusterfield to control the image resolution when saveLambda=TRUE and to clusterradius when expand is missing.
poisthresh	Numerical threshold below which the model will be treated as a Poisson process. See Details.
saveparents	Logical value indicating whether to save the locations of the parent points as an attribute.

Details

This algorithm generates a realisation of the ('modified') Thomas process, a special case of the Neyman-Scott process, inside the window win.

In the simplest case, where kappa and mu are single numbers, the algorithm generates a uniform Poisson point process of "parent" points with intensity kappa. Then each parent point is replaced by a random cluster of "offspring" points, the number of points per cluster being Poisson (mu) distributed, and their positions being isotropic Gaussian displacements from the cluster parent location. The resulting point pattern is a realisation of the classical "stationary Thomas process" generated inside the window win. This point process has intensity kappa * mu.

The algorithm can also generate spatially inhomogeneous versions of the Thomas process:

- The parent points can be spatially inhomogeneous. If the argument kappa is a function(x,y) or a pixel image (object of class "im"), then it is taken as specifying the intensity function of an inhomogeneous Poisson process that generates the parent points.
- The offspring points can be inhomogeneous. If the argument mu is a function(x,y) or a pixel image (object of class "im"), then it is interpreted as the reference density for offspring points, in the sense of Waagepetersen (2007). For a given parent point, the offspring constitute a Poisson process with intensity function equal to mu * f, where f is the Gaussian probability density centred at the parent point. Equivalently we first generate, for each parent point, a Poisson (mumax) random number of offspring (where M is the maximum value of mu) with independent Gaussian displacements from the parent location, and then randomly thin the offspring points, with retention probability mu/M.
- Both the parent points and the offspring points can be spatially inhomogeneous, as described above.

Note that if kappa is a pixel image, its domain must be larger than the window win. This is because an offspring point inside win could have its parent point lying outside win. In order to allow this, the simulation algorithm first expands the original window win by a distance expand and generates the Poisson process of parent points on this larger window. If kappa is a pixel image, its domain must contain this larger window.

The intensity of the Thomas process is kappa * mu if either kappa or mu is a single number. In the general case the intensity is an integral involving kappa, mu and f.

The Thomas process with homogeneous parents (i.e. where kappa is a single number) can be fitted to data using kppm. Currently it is not possible to fit the Thomas model with inhomogeneous parents.

If the pair correlation function of the model is very close to that of a Poisson process, deviating by less than poisthresh, then the model is approximately a Poisson process, and will be simulated as a Poisson process with intensity kappa * mu, using rpoispp. This avoids computations that would otherwise require huge amounts of memory.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Additionally, some intermediate results of the simulation are returned as attributes of this point pattern (see rNeymanScott). Furthermore, the simulated intensity function is returned as an attribute "Lambda", if saveLambda=TRUE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Diggle, P. J., Besag, J. and Gleaves, J. T. (1976) Statistical analysis of spatial point patterns by means of distance methods. *Biometrics* **32** 659–667.

Thomas, M. (1949) A generalisation of Poisson's binomial limit for use in ecology. *Biometrika* **36**, 18–25.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

rpoispp, rMatClust, rCauchy, rVarGamma, rNeymanScott, rGaussPoisson, kppm, clusterfit.

Examples

```
#homogeneous
X <- rThomas(10, 0.2, 5)
#inhomogeneous
Z <- as.im(function(x,y){ 5 * exp(2 * x - 1) }, owin())
Y <- rThomas(10, 0.2, Z)
```

---

run.simplepanel *Run Point-and-Click Interface*

Description

Execute various operations in a simple point-and-click user interface.

Usage

```
run.simplepanel(P, popup=TRUE, verbose = FALSE)
clear.simplepanel(P)
redraw.simplepanel(P, verbose = FALSE)
```

Arguments

P	An interaction panel (object of class "simplepanel", created by simplepanel or grow.simplepanel).
popup	Logical. If popup=TRUE (the default), the panel will be displayed in a new popup window. If popup=FALSE, the panel will be displayed on the current graphics window if it already exists, and on a new window otherwise.
verbose	Logical. If TRUE, debugging information will be printed.

Details

These commands enable the user to run a simple, robust, point-and-click interface to any R code. The interface is implemented using only the basic graphics package in R.

The argument P is an object of class "simplepanel", created by simplepanel or grow.simplepanel, which specifies the graphics to be displayed and the actions to be performed when the user interacts with the panel.

The command run.simplepanel(P) activates the panel: the display is initialised and the graphics system waits for the user to click the panel. While the panel is active, the user can only interact with the panel; the R command line interface and the R GUI cannot be used. When the panel terminates (typically because the user clicked a button labelled Exit), control returns to the R command line interface and the R GUI.

The command clear.simplepanel(P) clears all the display elements in the panel, resulting in a blank display except for the title of the panel.

The command redraw.simplepanel(P) redraws all the buttons of the panel, according to the redraw functions contained in the panel.

If popup=TRUE (the default), run.simplepanel begins by calling dev.new so that a new popup window is created; this window is closed using dev.off when run.simplepanel terminates. If popup=FALSE, the panel will be displayed on the current graphics window if it already exists, and on a new window otherwise; this window is not closed when run.simplepanel terminates.

For more sophisticated control of the graphics focus (for example, to use the panel to control the display on another window), initialise the graphics devices yourself using dev.new or similar commands; save these devices in the shared environment env of the panel P; and write the click/redraw functions of P in such a way that they access these devices using dev.set. Then use run.simplepanel with popup=FALSE.

Value

The return value of run.simplepanel(P) is the value returned by the exit function of P. See simplepanel.

The functions clear.simplepanel and redraw.simplepanel return NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

simplepanel

Examples

```
if(interactive()) {
  # make boxes (alternatively use layout.boxes())
  Bminus <- square(1)
  Bvalue <- shift(Bminus, c(1.2, 0))
  Bplus <- shift(Bvalue, c(1.2, 0))
  Bdone <- shift(Bplus, c(1.2, 0))
  myboxes <- list(Bminus, Bvalue, Bplus, Bdone)
  myB <- do.call(boundingbox,myboxes)

  # make environment containing an integer count
  myenv <- new.env()
  assign("answer", 0, envir=myenv)

  # what to do when finished: return the count.
  myexit <- function(e) { return(get("answer", envir=e)) }

  # button clicks
  # decrement the count
  Cminus <- function(e, xy) {
   ans <- get("answer", envir=e)
   assign("answer", ans - 1, envir=e)
   return(TRUE)
 }
 # display the count (clicking does nothing)
 Cvalue <- function(...) { TRUE }
 # increment the count
 Cplus <- function(e, xy) {
  ans <- get("answer", envir=e)
  assign("answer", ans + 1, envir=e)
  return(TRUE)
 }
 # quit button
 Cdone <- function(e, xy) { return(FALSE) }

 myclicks <- list("-"=Cminus,
                  value=Cvalue,
                  "+"=Cplus,
                  done=Cdone)

 # redraw the button that displays the current value of the count
 Rvalue <- function(button, nam, e) {
   plot(button, add=TRUE)
   ans <- get("answer", envir=e)
   text(centroid.owin(button), labels=ans)
   return(TRUE)
}

# make the panel
P <- simplepanel("Counter",
                 B=myB, boxes=myboxes,
                 clicks=myclicks,
```

```
                    redraws = list(NULL, Rvalue, NULL, NULL),
                    exit=myexit, env=myenv)
P

run.simplepanel(P)
}
```

runifdisc *Generate N Uniform Random Points in a Disc*

Description

Generate a random point pattern containing n independent uniform random points in a circular disc.

Usage

```
runifdisc(n, radius=1, centre=c(0,0), ..., nsim=1, drop=TRUE)
```

Arguments

n	Number of points.
radius	Radius of the circle.
centre	Coordinates of the centre of the circle.
...	Arguments passed to disc controlling the accuracy of approximation to the circle.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates n independent random points, uniformly distributed in a circular disc.

It is faster (for a circular window) than the general code used in runifpoint.

To generate random points in an ellipse, first generate points in a circle using runifdisc, then transform to an ellipse using affine, as shown in the examples.

To generate random points in other windows, use runifpoint. To generate non-uniform random points, use rpoint.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

disc, runifpoint, rpoint

Examples

```
# 100 random points in the unit disc
plot(runifdisc(100))
# 42 random points in the ellipse with major axis 3 and minor axis 1
X <- runifdisc(42)
Y <- affine(X, mat=diag(c(3,1)))
plot(Y)
```

---

runiflpp *Uniform Random Points on a Linear Network*

Description

Generates n random points, independently and uniformly distributed, on a linear network.

Usage

```
runiflpp(n, L, nsim=1, drop=TRUE)
```

Arguments

n	Number of random points to generate. A nonnegative integer, or a vector of integers specifying the number of points of each type.
L	A linear network (object of class "linnet", see linnet).
nsim	Number of simulated realisations to generate.
drop	Logical value indicating what to do when nsim=1. If drop=TRUE (the default), the result is a point pattern. If drop=FALSE, the result is a list with one entry which is a point pattern.

Details

This function uses runifpointOnLines to generate the random points.

Value

If nsim = 1 and drop=TRUE, a point pattern on the linear network, i.e.\ an object of class "lpp". Otherwise, a list of such point patterns.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

rlpp for non-uniform random points; rpoislpp for Poisson point process;

lpp, linnet

Examples

```
data(simplenet)
X <- runiflpp(10, simplenet)
plot(X)
# marked
Z <- runiflpp(c(a=10, b=3), simplenet)
```

---

runifpoint *Generate N Uniform Random Points*

Description

Generate a random point pattern containing n independent uniform random points.

Usage

```
runifpoint(n, win=owin(c(0,1),c(0,1)), giveup=1000, warn=TRUE, ...,
           nsim=1, drop=TRUE, ex=NULL)
```

Arguments

n	Number of points.
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin.
giveup	Number of attempts in the rejection method after which the algorithm should stop trying to generate new points.
warn	Logical. Whether to issue a warning if n is very large. See Details.
...	Ignored.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.
ex	Optional. A point pattern to use as the example. If ex is given and n and win are missing, then n and win will be calculated from the point pattern ex.

Details

This function generates n independent random points, uniformly distributed in the window win. (For nonuniform distributions, see rpoint.)

The algorithm depends on the type of window, as follows:

- If win is a rectangle then n independent random points, uniformly distributed in the rectangle, are generated by assigning uniform random values to their cartesian coordinates.
- If win is a binary image mask, then a random sequence of pixels is selected (using sample) with equal probabilities. Then for each pixel in the sequence we generate a uniformly distributed random point in that pixel.
- If win is a polygonal window, the algorithm uses the rejection method. It finds a rectangle enclosing the window, generates points in this rectangle, and tests whether they fall in the desired window. It gives up when giveup * n tests have been performed without yielding n successes.

The algorithm for binary image masks is faster than the rejection method but involves discretisation.

If `warn=TRUE`, then a warning will be issued if `n` is very large. The threshold is `spatstat.options("huge.npoints")`. This warning has no consequences, but it helps to trap a number of common errors.

Value

A point pattern (an object of class `"ppp"`) if `nsim=1`, or a list of point patterns if `nsim > 1`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`ppp.object`, `owin.object`, `rpoispp`, `rpoint`

Examples

```
# 100 random points in the unit square
pp <- runifpoint(100)
# irregular window
data(letterR)
# polygonal
pp <- runifpoint(100, letterR)
# binary image mask
pp <- runifpoint(100, as.mask(letterR))
##
# randomising an existing point pattern
runifpoint(npoints(cells), win=Window(cells))
runifpoint(ex=cells)
```

`runifpoint3` *Generate N Uniform Random Points in Three Dimensions*

Description

Generate a random point pattern containing `n` independent, uniform random points in three dimensions.

Usage

```
runifpoint3(n, domain = box3(), nsim=1, drop=TRUE)
```

Arguments

`n`	Number of points to be generated.
`domain`	Three-dimensional box in which the process should be generated. An object of class `"box3"`.
`nsim`	Number of simulated realisations to be generated.
`drop`	Logical. If `nsim=1` and `drop=TRUE` (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function generates n independent random points, uniformly distributed in the three-dimensional box domain.

Value

If nsim = 1 and drop=TRUE, a point pattern in three dimensions (an object of class "pp3"). If nsim > 1, a list of such point patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rpoispp3, pp3, box3

Examples

```
X <- runifpoint3(50)
```

---

runifpointOnLines *Generate N Uniform Random Points On Line Segments*

Description

Given a line segment pattern, generate a random point pattern consisting of n points uniformly distributed on the line segments.

Usage

```
runifpointOnLines(n, L, nsim=1)
```

Arguments

n	Number of points to generate.
L	Line segment pattern (object of class "psp") on which the points should lie.
nsim	Number of simulated realisations to be generated.

Details

This command generates a point pattern consisting of n independent random points, each point uniformly distributed on the line segment pattern. This means that, for each random point,

- the probability of falling on a particular segment is proportional to the length of the segment; and
- given that the point falls on a particular segment, it has uniform probability density along that segment.

If n is a single integer, the result is an unmarked point pattern containing n points. If n is a vector of integers, the result is a marked point pattern, with m different types of points, where m = length(n), in which there are n[j] points of type j.

Value

If `nsim = 1`, a point pattern (object of class "ppp") with the same window as `L`. If `nsim > 1`, a list of point patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

psp, ppp, pointsOnLines, runifpoint

Examples

```
X <- psp(runif(10), runif(10), runif(10), runif(10),  window=owin())
Y <- runifpointOnLines(20, X)
plot(X, main="")
plot(Y, add=TRUE)
Z <- runifpointOnLines(c(5,5), X)
```

---

runifpointx — *Generate N Uniform Random Points in Any Dimensions*

Description

Generate a random point pattern containing `n` independent, uniform random points in any number of spatial dimensions.

Usage

```
runifpointx(n, domain, nsim=1, drop=TRUE)
```

Arguments

n	Number of points to be generated.
domain	Multi-dimensional box in which the process should be generated. An object of class "boxx".
nsim	Number of simulated realisations to be generated.
drop	Logical. If `nsim=1` and `drop=TRUE` (the default), the result will be a point pattern, rather than a list containing a single point pattern.

Details

This function generates a pattern of `n` independent random points, uniformly distributed in the multi-dimensional box `domain`.

Value

If `nsim = 1` and `drop=TRUE`, a point pattern (an object of class "ppx"). If `nsim > 1` or `drop=FALSE`, a list of such point patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rpoisppx, ppx, boxx

Examples

```
w <- boxx(x=c(0,1), y=c(0,1), z=c(0,1), t=c(0,3))
X <- runifpointx(50, w)
```

---

rVarGamma *Simulate Neyman-Scott Point Process with Variance Gamma cluster kernel*

Description

Generate a random point pattern, a simulated realisation of the Neyman-Scott process with Variance Gamma (Bessel) cluster kernel.

Usage

```
rVarGamma(kappa, nu, scale, mu, win = owin(),
          thresh = 0.001, nsim=1, drop=TRUE,
          saveLambda=FALSE, expand = NULL, ...,
          poisthresh=1e-6, saveparents=TRUE)
```

Arguments

kappa	Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.
nu	Shape parameter for the cluster kernel. A number greater than -1.
scale	Scale parameter for cluster kernel. Determines the size of clusters. A positive number in the same units as the spatial coordinates.
mu	Mean number of points per cluster (a single positive number) or reference intensity for the cluster points (a function or a pixel image).
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin.
thresh	Threshold relative to the cluster kernel value at the origin (parent location) determining when the cluster kernel will be treated as zero for simulation purposes. Will be overridden by argument expand if that is given.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.
saveLambda	Logical. If TRUE then the random intensity corresponding to the simulated parent points will also be calculated and saved, and returns as an attribute of the point pattern.

expand	Numeric. Size of window expansion for generation of parent points. By default determined by calling clusterradius with the numeric threshold value given in thresh.
...	Passed to clusterfield to control the image resolution when saveLambda=TRUE and to clusterradius when expand is missing or NULL.
poisthresh	Numerical threshold below which the model will be treated as a Poisson process. See Details.
saveparents	Logical value indicating whether to save the locations of the parent points as an attribute.

Details

This algorithm generates a realisation of the Neyman-Scott process with Variance Gamma (Bessel) cluster kernel, inside the window win.

The process is constructed by first generating a Poisson point process of "parent" points with intensity kappa. Then each parent point is replaced by a random cluster of points, the number of points in each cluster being random with a Poisson (mu) distribution, and the points being placed independently and uniformly according to a Variance Gamma kernel.

The shape of the kernel is determined by the dimensionless index nu. This is the parameter $\nu' = \alpha/2 - 1$ appearing in equation (12) on page 126 of Jalilian et al (2013).

The scale of the kernel is determined by the argument scale, which is the parameter η appearing in equations (12) and (13) of Jalilian et al (2013). It is expressed in units of length (the same as the unit of length for the window win).

In this implementation, parent points are not restricted to lie in the window; the parent process is effectively the uniform Poisson process on the infinite plane.

This model can be fitted to data by the method of minimum contrast, maximum composite likelihood or Palm likelihood using kppm.

The algorithm can also generate spatially inhomogeneous versions of the cluster process:

- The parent points can be spatially inhomogeneous. If the argument kappa is a function(x,y) or a pixel image (object of class "im"), then it is taken as specifying the intensity function of an inhomogeneous Poisson process that generates the parent points.
- The offspring points can be inhomogeneous. If the argument mu is a function(x,y) or a pixel image (object of class "im"), then it is interpreted as the reference density for offspring points, in the sense of Waagepetersen (2006).

When the parents are homogeneous (kappa is a single number) and the offspring are inhomogeneous (mu is a function or pixel image), the model can be fitted to data using kppm, or using vargamma.estK or vargamma.estpcf applied to the inhomogeneous K function.

If the pair correlation function of the model is very close to that of a Poisson process, deviating by less than poisthresh, then the model is approximately a Poisson process, and will be simulated as a Poisson process with intensity kappa * mu, using rpoispp. This avoids computations that would otherwise require huge amounts of memory.

Value

A point pattern (an object of class "ppp") if nsim=1, or a list of point patterns if nsim > 1.

Additionally, some intermediate results of the simulation are returned as attributes of this point pattern (see rNeymanScott). Furthermore, the simulated intensity function is returned as an attribute "Lambda", if saveLambda=TRUE.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.ed

References

Jalilian, A., Guan, Y. and Waagepetersen, R. (2013) Decomposition of variance for spatial Cox processes. *Scandinavian Journal of Statistics* **40**, 119-137.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

rpoispp, rNeymanScott, kppm.

vargamma.estK, vargamma.estpcf.

Examples

```
# homogeneous
X <- rVarGamma(30, 2, 0.02, 5)
# inhomogeneous
ff <- function(x,y){ exp(2 - 3 * abs(x)) }
Z <- as.im(ff, W= owin())
Y <- rVarGamma(30, 2, 0.02, Z)
YY <- rVarGamma(ff, 2, 0.02, 3)
```

SatPiece *Piecewise Constant Saturated Pairwise Interaction Point Process Model*

Description

Creates an instance of a saturated pairwise interaction point process model with piecewise constant potential function. The model can then be fitted to point pattern data.

Usage

```
SatPiece(r, sat)
```

Arguments

r	vector of jump points for the potential function
sat	vector of saturation values, or a single saturation value

Details

This is a generalisation of the Geyer saturation point process model, described in Geyer, to the case of multiple interaction distances. It can also be described as the saturated analogue of a pairwise interaction process with piecewise-constant pair potential, described in PairPiece.

The saturated point process with interaction radii $r_1, \ldots, r_k$, saturation thresholds $s_1, \ldots, s_k$, intensity parameter β and interaction parameters $\gamma_1, \ldots, gamma_k$, is the point process in which each point x_i in the pattern X contributes a factor

$$\beta \gamma_1^{v_1(x_i, X)} \ldots gamma_k^{v_k(x_i, X)}$$

to the probability density of the point pattern, where

$$v_j(x_i, X) = \min(s_j, t_j(x_i, X))$$

where $t_j(x_i, X)$ denotes the number of points in the pattern X which lie at a distance between r_{j-1} and r_j from the point x_i. We take $r_0 = 0$ so that $t_1(x_i, X)$ is the number of points of X that lie within a distance r_1 of the point x_i.

`SatPiece` is used to fit this model to data. The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the piecewise constant Saturated pairwise interaction is yielded by the function `SatPiece()`. See the examples below.

Simulation of this point process model is not yet implemented. This model is not locally stable (the conditional intensity is unbounded).

The argument `r` specifies the vector of interaction distances. The entries of `r` must be strictly increasing, positive numbers.

The argument `sat` specifies the vector of saturation parameters. It should be a vector of the same length as `r`, and its entries should be nonnegative numbers. Thus `sat[1]` corresponds to the distance range from `0` to `r[1]`, and `sat[2]` to the distance range from `r[1]` to `r[2]`, etc. Alternatively `sat` may be a single number, and this saturation value will be applied to every distance range.

Infinite values of the saturation parameters are also permitted; in this case $v_j(x_i, X) = t_j(x_i, X)$ and there is effectively no 'saturation' for the distance range in question. If all the saturation parameters are set to `Inf` then the model is effectively a pairwise interaction process, equivalent to `PairPiece` (however the interaction parameters γ obtained from `SatPiece` are the square roots of the parameters γ obtained from `PairPiece`).

If `r` is a single number, this model is virtually equivalent to the Geyer process, see `Geyer`.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

in collaboration with Hao Wang and Jeff Picka

See Also

`ppm`, `pairsat.family`, `Geyer`, `PairPiece`, `BadGey`.

Examples

```
SatPiece(c(0.1,0.2), c(1,1))
# prints a sensible description of itself
SatPiece(c(0.1,0.2), 1)
data(cells)
```

```
    ppm(cells, ~1, SatPiece(c(0.07, 0.1, 0.13), 2))
    # fit a stationary piecewise constant Saturated pairwise interaction process

    ## Not run:
    ppm(cells, ~polynom(x,y,3), SatPiece(c(0.07, 0.1, 0.13), 2))
    # nonstationary process with log-cubic polynomial trend

## End(Not run)
```

`Saturated` *Saturated Pairwise Interaction model*

Description

Experimental.

Usage

```
Saturated(pot, name)
```

Arguments

`pot`	An S language function giving the user-supplied pairwise interaction potential.
`name`	Character string.

Details

This is experimental. It constructs a member of the "saturated pairwise" family `pairsat.family`.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process.

Author(s)

Adrian Baddeley <`Adrian.Baddeley@curtin.edu.au`>

and Rolf Turner <`r.turner@auckland.ac.nz`>

See Also

`ppm`, `pairsat.family`, `Geyer`, `SatPiece`, `ppm.object` scalardilate *Apply Scalar Dilation*

Description

Applies scalar dilation to a plane geometrical object, such as a point pattern or a window, relative to a specified origin.

Usage

```
scalardilate(X, f, ...)

## S3 method for class 'im'
scalardilate(X, f, ..., origin=NULL)

## S3 method for class 'owin'
scalardilate(X, f, ..., origin=NULL)

## S3 method for class 'ppp'
scalardilate(X, f, ..., origin=NULL)

## S3 method for class 'psp'
scalardilate(X, f, ..., origin=NULL)

## Default S3 method:
scalardilate(X, f, ...)
```

Arguments

X	Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), a window (object of class "owin"), a pixel image (class "im") and so on.
f	Scalar dilation factor. A finite number greater than zero.
...	Ignored by the methods.
origin	Origin for the scalar dilation. Either a vector of 2 numbers, or one of the character strings "centroid", "midpoint" or "bottomleft" (partially matched).

Details

This command performs scalar dilation of the object `X` by the factor `f` relative to the origin specified by `origin`.

The function `scalardilate` is generic, with methods for windows (class "owin"), point patterns (class "ppp"), pixel images (class "im"), line segment patterns (class "psp") and a default method.

If the argument `origin` is not given, then every spatial coordinate is multiplied by the factor `f`.

If `origin` is given, then scalar dilation is performed relative to the specified origin. Effectively, `X` is shifted so that `origin` is moved to `c(0,0)`, then scalar dilation is performed, then the result is shifted so that `c(0,0)` is moved to `origin`.

This command is a special case of an affine transformation: see `affine`.

Value

Another object of the same type, representing the result of applying the scalar dilation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

affine, shift

Examples

```
plot(letterR)
plot(scalardilate(letterR, 0.7, origin="bot"), col="red", add=TRUE)
```

---

scaletointerval *Rescale Data to Lie Between Specified Limits*

Description

Rescales a dataset so that the values range exactly between the specified limits.

Usage

```
  scaletointerval(x, from=0, to=1, xrange=range(x))
  ## Default S3 method:
scaletointerval(x, from=0, to=1, xrange=range(x))
  ## S3 method for class 'im'
scaletointerval(x, from=0, to=1, xrange=range(x))
```

Arguments

x	Data to be rescaled.
from,to	Lower and upper endpoints of the interval to which the values of x should be rescaled.
xrange	Optional range of values of x that should be mapped to the new interval.

Details

These functions rescale a dataset x so that its values range exactly between the limits from and to.

The method for pixel images (objects of class "im") applies this scaling to the pixel values of x.

Rescaling cannot be performed if the values in x are not interpretable as numeric, or if the values in x are all equal.

Value

An object of the same type as x.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

scale

Examples

```
X <- as.im(function(x,y) {x+y+3}, unit.square())
summary(X)
Y <- scaletointerval(X)
summary(Y)
```

---

scan.test *Spatial Scan Test*

Description

Performs the Spatial Scan Test for clustering in a spatial point pattern, or for clustering of one type of point in a bivariate spatial point pattern.

Usage

```
scan.test(X, r, ...,
          method = c("poisson", "binomial"),
          nsim = 19,
          baseline = NULL,
          case = 2,
          alternative = c("greater", "less", "two.sided"),
          verbose = TRUE)
```

Arguments

X	A point pattern (object of class "ppp").
r	Radius of circle to use. A single number or a numeric vector.
...	Optional. Arguments passed to as.mask to determine the spatial resolution of the computations.
method	Either "poisson" or "binomial" specifying the type of likelihood.
nsim	Number of simulations for computing Monte Carlo p-value.
baseline	Baseline for the Poisson intensity, if method="poisson". A pixel image or a function.
case	Which type of point should be interpreted as a case, if method="binomial". Integer or character string.
alternative	Alternative hypothesis: "greater" if the alternative postulates that the mean number of points inside the circle will be greater than expected under the null.
verbose	Logical. Whether to print progress reports.

Details

The spatial scan test (Kulldorf, 1997) is applied to the point pattern X.

In a nutshell,

- If method="poisson" then a significant result would mean that there is a circle of radius r, located somewhere in the spatial domain of the data, which contains a significantly higher than expected number of points of X. That is, the pattern X exhibits spatial clustering.
- If method="binomial" then X must be a bivariate (two-type) point pattern. By default, the first type of point is interpreted as a control (non-event) and the second type of point as a case (event). A significant result would mean that there is a circle of radius r which contains a significantly higher than expected number of cases. That is, the cases are clustered together, conditional on the locations of all points.

Following is a more detailed explanation.

- If method="poisson" then the scan test based on Poisson likelihood is performed (Kulldorf, 1997). The dataset X is treated as an unmarked point pattern. By default (if baseline is not specified) the null hypothesis is complete spatial randomness CSR (i.e. a uniform Poisson process). The alternative hypothesis is a Poisson process with one intensity β_1 inside some circle of radius r and another intensity β_0 outside the circle. If baseline is given, then it should be a pixel image or a function(x,y). The null hypothesis is an inhomogeneous Poisson process with intensity proportional to baseline. The alternative hypothesis is an inhomogeneous Poisson process with intensity beta1 * baseline inside some circle of radius r, and beta0 * baseline outside the circle.
- If method="binomial" then the scan test based on binomial likelihood is performed (Kulldorf, 1997). The dataset X must be a bivariate point pattern, i.e. a multitype point pattern with two types. The null hypothesis is that all permutations of the type labels are equally likely. The alternative hypothesis is that some circle of radius r has a higher proportion of points of the second type, than expected under the null hypothesis.

The result of scan.test is a hypothesis test (object of class "htest") which can be plotted to report the results. The component p.value contains the p-value.

The result of scan.test can also be plotted (using the plot method for the class "scan.test"). The plot is a pixel image of the Likelihood Ratio Test Statistic (2 times the log likelihood ratio) as a function of the location of the centre of the circle. This pixel image can be extracted from the object using as.im.scan.test. The Likelihood Ratio Test Statistic is computed by scanLRTS.

Value

An object of class "htest" (hypothesis test) which also belongs to the class "scan.test". Printing this object gives the result of the test. Plotting this object displays the Likelihood Ratio Test Statistic as a function of the location of the centre of the circle.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Kulldorff, M. (1997) A spatial scan statistic. *Communications in Statistics — Theory and Methods* **26**, 1481–1496.

See Also

plot.scan.test, as.im.scan.test, relrisk, scanLRTS

Examples

```
nsim <- if(interactive()) 19 else 2
rr <- if(interactive()) seq(0.5, 1, by=0.1) else c(0.5, 1)
scan.test(redwood, 0.1 * rr, method="poisson", nsim=nsim)
scan.test(chorley, rr, method="binomial", case="larynx", nsim=nsim)
```

---

scanLRTS *Likelihood Ratio Test Statistic for Scan Test*

Description

Calculate the Likelihood Ratio Test Statistic for the Scan Test, at each spatial location.

Usage

```
scanLRTS(X, r, ...,
   method = c("poisson", "binomial"),
   baseline = NULL, case = 2,
   alternative = c("greater", "less", "two.sided"),
   saveopt = FALSE,
   Xmask = NULL)
```

Arguments

X	A point pattern (object of class "ppp").
r	Radius of circle to use. A single number or a numeric vector.
...	Optional. Arguments passed to as.mask to determine the spatial resolution of the computations.
method	Either "poisson" or "binomial" specifying the type of likelihood.
baseline	Baseline for the Poisson intensity, if method="poisson". A pixel image or a function.
case	Which type of point should be interpreted as a case, if method="binomial". Integer or character string.
alternative	Alternative hypothesis: "greater" if the alternative postulates that the mean number of points inside the circle will be greater than expected under the null.
saveopt	Logical value indicating to save the optimal value of r at each location.
Xmask	Internal use only.

Details

This command computes, for all spatial locations u, the Likelihood Ratio Test Statistic $\Lambda(u)$ for a test of homogeneity at the location u, as described below. The result is a pixel image giving the values of $\Lambda(u)$ at each pixel.

The **maximum** value of $\Lambda(u)$ over all locations u is the *scan statistic*, which is the basis of the *scan test* performed by scan.test.

- If method="poisson" then the test statistic is based on Poisson likelihood. The dataset X is treated as an unmarked point pattern. By default (if baseline is not specified) the null hypothesis is complete spatial randomness CSR (i.e. a uniform Poisson process). At the spatial location u, the alternative hypothesis is a Poisson process with one intensity β_1 inside the circle of radius r centred at u, and another intensity β_0 outside the circle. If baseline is given, then it should be a pixel image or a function(x,y). The null hypothesis is an inhomogeneous Poisson process with intensity proportional to baseline. The alternative hypothesis is an inhomogeneous Poisson process with intensity beta1 * baseline inside the circle, and beta0 * baseline outside the circle.
- If method="binomial" then the test statistic is based on binomial likelihood. The dataset X must be a bivariate point pattern, i.e. a multitype point pattern with two types. The null hypothesis is that all permutations of the type labels are equally likely. The alternative hypothesis is that the circle of radius r centred at u has a higher proportion of points of the second type, than expected under the null hypothesis.

If r is a vector of more than one value for the radius, then the calculations described above are performed for every value of r. Then the maximum over r is taken for each spatial location u. The resulting pixel value of scanLRTS at a location u is the profile maximum of the Likelihood Ratio Test Statistic, that is, the maximum of the Likelihood Ratio Test Statistic for circles of all radii, centred at the same location u.

If you have already performed a scan test using scan.test, the Likelihood Ratio Test Statistic can be extracted from the test result using the function as.im.scan.test.

Value

A pixel image (object of class "im") whose pixel values are the values of the (profile) Likelihood Ratio Test Statistic at each spatial location.

Warning: window size

Note that the result of scanLRTS is a pixel image on a larger window than the original window of X. The expanded window contains the centre of any circle of radius r that has nonempty intersection with the original window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Kulldorff, M. (1997) A spatial scan statistic. *Communications in Statistics — Theory and Methods* **26**, 1481–1496.

See Also

scan.test, as.im.scan.test

Examples

```
plot(scanLRTS(redwood, 0.1, method="poisson"))
sc <- scanLRTS(chorley, 1, method="binomial", case="larynx")
plot(sc)
scanstatchorley <- max(sc)
```

---

scanpp — *Read Point Pattern From Data File*

Description

Reads a point pattern dataset from a text file.

Usage

```
scanpp(filename, window, header=TRUE, dir="", factor.marks=NULL, ...)
```

Arguments

filename	String name of the file containing the coordinates of the points in the point pattern, and their marks if any.
window	Window for the point pattern. An object of class "owin".
header	Logical flag indicating whether the first line of the file contains headings for the columns. Passed to read.table.
dir	String containing the path name of the directory in which filename is to be found. Default is the current directory.
factor.marks	Logical vector (or NULL) indicating whether marks are to be interpreted as factors. Defaults to NULL which means that strings will be interpreted as factors while numeric variables will not. See details.
...	Ignored.

Details

This simple function reads a point pattern dataset from a file containing the cartesian coordinates of its points, and optionally the mark values for these points.

The file identified by filename in directory dir should be a text file that can be read using read.table. Thus, each line of the file (except possibly the first line) contains data for one point in the point pattern. Data are arranged in columns. There should be either two columns (for an unmarked point pattern) or more columns (for a marked point pattern).

If header=FALSE then the first two columns of data will be interpreted as the x and y coordinates of points. Remaining columns, if present, will be interpreted as containing the marks for these points.

If header=TRUE then the first line of the file should contain string names for each of the columns of data. If there are columns named x and y then these will be taken as the cartesian coordinates, and any remaining columns will be taken as the marks. If there are no columns named x and y then the first and second columns will be taken as the cartesian coordinates.

If a logical vector is provided for `factor.marks` the length should equal the number of mark columns (a shorter `factor.marks` is recycled to this length). This vector is then used to determine which mark columns should be interpreted as factors. Note: Strings will not be interpreted as factors if the corresponding entry in `factor.marks` is `FALSE`.

Note that there is intentionally no default for `window`. The window of observation should be specified. If you really need to estimate the window, use the Ripley-Rasson estimator `ripras`.

Value

A point pattern (an object of class `"ppp"`, see `ppp.object`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppp.object`, `ppp`, `as.ppp`, `ripras`

`sdr` *Sufficient Dimension Reduction*

Description

Given a point pattern and a set of predictors, find a minimal set of new predictors, each constructed as a linear combination of the original predictors.

Usage

```
sdr(X, covariates, method = c("DR", "NNIR", "SAVE", "SIR", "TSE"),
    Dim1 = 1, Dim2 = 1, predict=FALSE)
```

Arguments

`X`	A point pattern (object of class `"ppp"`).
`covariates`	A list of pixel images (objects of class `"im"`) to serve as predictor variables.
`method`	Character string indicating which method to use. See Details.
`Dim1`	Dimension of the first order Central Intensity Subspace (applicable when `method` is `"DR"`, `"NNIR"`, `"SAVE"` or `"TSE"`).
`Dim2`	Dimension of the second order Central Intensity Subspace (applicable when `method="TSE"`).
`predict`	Logical value indicating whether to compute the new predictors as well.

Details

Given a point pattern X and predictor variables $Z_1, \ldots, Z_p$, Sufficient Dimension Reduction methods (Guan and Wang, 2010) attempt to find a minimal set of new predictor variables, each constructed by taking a linear combination of the original predictors, which explain the dependence of X on $Z_1, \ldots, Z_p$. The methods do not assume any particular form of dependence of the point pattern on the predictors. The predictors are assumed to be Gaussian random fields.

Available methods are:

`method="DR"`	directional regression
`method="NNIR"`	nearest neighbour inverse regression

`method="SAVE"` & sliced average variance estimation
`method="SIR"` & sliced inverse regression
`method="TSE"` & two-step estimation

The result includes a matrix `B` whose columns are estimates of the basis vectors of the space of new predictors. That is, the `j`th column of `B` expresses the `j`th new predictor as a linear combination of the original predictors.

If `predict=TRUE`, the new predictors are also evaluated. They can also be evaluated using `sdrPredict`.

Value

A list with components `B, M` or `B, M1, M2` where `B` is a matrix whose columns are estimates of the basis vectors for the space, and `M` or `M1,M2` are matrices containing estimates of the kernel.

If `predict=TRUE`, the result also includes a component `Y` which is a list of pixel images giving the values of the new predictors.

Author(s)

Matlab original by Yongtao Guan, translated to R by Suman Rakshit.

References

Guan, Y. and Wang, H. (2010) Sufficient dimension reduction for spatial point processes directed by Gaussian random fields. *Journal of the Royal Statistical Society, Series B*, **72**, 367–387.

See Also

`sdrPredict` to compute the new predictors from the coefficient matrix.

`dimhat` to estimate the subspace dimension.

`subspaceDistance`

Examples

```
A <- sdr(bei, bei.extra, predict=TRUE)
A
Y1 <- A$Y[[1]]
plot(Y1)
points(bei, pch=".", cex=2)
# investigate likely form of dependence
plot(rhohat(bei, Y1))
```

sdrPredict *Compute Predictors from Sufficient Dimension Reduction*

Description

Given the result of a Sufficient Dimension Reduction method, compute the new predictors.

Usage

```
sdrPredict(covariates, B)
```

Arguments

covariates	A list of pixel images (objects of class "im").
B	Either a matrix of coefficients for the covariates, or the result of a call to sdr.

Details

This function assumes that sdr has already been used to find a minimal set of predictors based on the covariates. The argument B should be either the result of sdr or the coefficient matrix returned as one of the results of sdr. The columns of this matrix define linear combinations of the covariates. This function evaluates those linear combinations, and returns a list of pixel images containing the new predictors.

Value

A list of pixel images (objects of class "im") with one entry for each column of B.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

sdr

Examples

```
A <- sdr(bei, bei.extra)
Y <- sdrPredict(bei.extra, A)
Y
```

segregation.test *Test of Spatial Segregation of Types*

Description

Performs a Monte Carlo test of spatial segregation of the types in a multitype point pattern.

Usage

```
segregation.test(X, ...)

## S3 method for class 'ppp'
segregation.test(X, ..., nsim = 19,
       permute = TRUE, verbose = TRUE, Xname)
```

Arguments

X	Multitype point pattern (object of class "ppp" with factor-valued marks).
...	Additional arguments passed to relrisk.ppp to control the smoothing parameter or bandwidth selection.
nsim	Number of simulations for the Monte Carlo test.
permute	Argument passed to rlabel. If TRUE (the default), randomisation is performed by randomly permuting the labels of X. If FALSE, randomisation is performing by resampling the labels with replacement.
verbose	Logical value indicating whether to print progress reports.
Xname	Optional character string giving the name of the dataset X.

Details

The Monte Carlo test of spatial segregation of types, proposed by Kelsall and Diggle (1995) and Diggle et al (2005), is applied to the point pattern X. The test statistic is

$$T = \sum_i \sum_m \left(\widehat{p}(m \mid x_i) - \overline{p}_m\right)^2$$

where $\widehat{p}(m \mid x_i)$ is the leave-one-out kernel smoothing estimate of the probability that the i-th data point has type m, and $\overline{p}_m$ is the average fraction of data points which are of type m. The statistic T is evaluated for the data and for nsim randomised versions of X, generated by randomly permuting or resampling the marks.

Note that, by default, automatic bandwidth selection will be performed separately for each randomised pattern. This computation can be very time-consuming but is necessary for the test to be valid in most conditions. A short-cut is to specify the value of the smoothing bandwidth sigma as shown in the examples.

Value

An object of class "htest" representing the result of the test.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Kelsall, J.E. and Diggle, P.J. (1995) Kernel estimation of relative risk. *Bernoulli* **1**, 3–16.

Diggle, P.J., Zheng, P. and Durr, P. (2005) Non-parametric estimation of spatial segregation in a multivariate point process: bovine tuberculosis in Cornwall, UK. *Applied Statistics* **54**, 645–658.

See Also

relrisk

Examples

```
segregation.test(hyytiala, 5)

if(interactive()) segregation.test(hyytiala, hmin=0.05)
```

---

selfcrossing.psp *Crossing Points in a Line Segment Pattern*

Description

Finds any crossing points between the line segments in a line segment pattern.

Usage

```
selfcrossing.psp(A)
```

Arguments

A	Line segment pattern (object of class "psp").

Details

This function finds any crossing points between different line segments in the line segment pattern A.

A crossing point occurs whenever one of the line segments in A intersects another line segment in A, at a nonzero angle of intersection.

Value

Point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

crossing.psp, psp.object, ppp.object.

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(a, col="green", main="selfcrossing.psp")
P <- selfcrossing.psp(a)
plot(P, add=TRUE, col="red")
```

---

selfcut.psp *Cut Line Segments Where They Intersect*

Description

Finds any crossing points between the line segments in a line segment pattern, and cuts the segments into pieces at these crossing-points.

Usage

```
selfcut.psp(A, ..., eps)
```

Arguments

A	Line segment pattern (object of class "psp").
eps	Optional. Smallest permissible length of the resulting line segments. There is a sensible default.
...	Ignored.

Details

This function finds any crossing points between different line segments in the line segment pattern A, and cuts the line segments into pieces at these intersection points.

A crossing point occurs whenever one of the line segments in A intersects another line segment in A, at a nonzero angle of intersection.

Value

Another line segment pattern (object of class "psp") in the same window as A with the same kind of marks as A.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

selfcrossing.psp

Examples

```
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
Y <- selfcut.psp(X)
n <- nsegments(Y)
plot(Y %mark% factor(sample(seq_len(n), n, replace=TRUE)))
```

---

sessionLibs *Print Names and Version Numbers of Libraries Loaded*

Description

Prints the names and version numbers of libraries currently loaded by the user.

Usage

```
sessionLibs()
```

Details

This function prints a list of the libraries loaded by the user in the current session, giving just their name and version number. It obtains this information from `sessionInfo`.

This function is not needed in an interactive R session because the package startup messages will usually provide this information.

Its main use is in an `Sweave` script, where it is needed because the package startup messages are not printed.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`sessionInfo`

setcov *Set Covariance of a Window*

Description

Computes the set covariance function of a window.

Usage

```
setcov(W, V=W, ...)
```

Arguments

W	A window (object of class "owin".
V	Optional. Another window.
...	Optional arguments passed to as.mask to control the pixel resolution.

Details

The set covariance function of a region W in the plane is the function $C(v)$ defined for each vector v as the area of the intersection between W and $W + v$, where $W + v$ is the set obtained by shifting (translating) W by v.

We may interpret $C(v)$ as the area of the set of all points x in W such that $x + v$ also lies in W.

This command computes a discretised approximation to the set covariance function of any plane region W represented as a window object (of class "owin", see owin.object). The return value is a pixel image (object of class "im") whose greyscale values are values of the set covariance function.

The set covariance is computed using the Fast Fourier Transform, unless W is a rectangle, when an exact formula is used.

If the argument V is present, then setcov(W,V) computes the set *cross-covariance* function $C(x)$ defined for each vector x as the area of the intersection between W and $V + x$.

Value

A pixel image (an object of class "im") representing the set covariance function of W, or the cross-covariance of W and V.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

imcov, owin, as.owin, erosion

Examples

```
w <- owin(c(0,1),c(0,1))
v <- setcov(w)
plot(v)
```

sharpen *Data Sharpening of Point Pattern*

Description

Performs Choi-Hall data sharpening of a spatial point pattern.

Usage

```
sharpen(X, ...)
## S3 method for class 'ppp'
sharpen(X, sigma=NULL, ...,
                      varcov=NULL, edgecorrect=FALSE)
```

Arguments

`X`	A marked point pattern (object of class "ppp").
`sigma`	Standard deviation of isotropic Gaussian smoothing kernel.
`varcov`	Variance-covariance matrix of anisotropic Gaussian kernel. Incompatible with `sigma`.
`edgecorrect`	Logical value indicating whether to apply edge effect bias correction.
`...`	Arguments passed to `density.ppp` to control the pixel resolution of the result.

Details

Choi and Hall (2001) proposed a procedure for *data sharpening* of spatial point patterns. This procedure is appropriate for earthquake epicentres and other point patterns which are believed to exhibit strong concentrations of points along a curve. Data sharpening causes such points to concentrate more tightly along the curve.

If the original data points are $X_1, \ldots, X_n$ then the sharpened points are

$$\hat{X}_i = \frac{\sum_j X_j k(X_j - X_i)}{\sum_j k(X_j - X_i)}$$

where k is a smoothing kernel in two dimensions. Thus, the new point $\hat{X}_i$ is a vector average of the nearby points $X[j]$.

The function `sharpen` is generic. It currently has only one method, for two-dimensional point patterns (objects of class "ppp").

If `sigma` is given, the smoothing kernel is the isotropic two-dimensional Gaussian density with standard deviation `sigma` in each axis. If `varcov` is given, the smoothing kernel is the Gaussian density with variance-covariance matrix `varcov`.

The data sharpening procedure tends to cause the point pattern to contract away from the boundary of the window. That is, points `X_i`X[i] that lie 'quite close to the edge of the window of the point pattern tend to be displaced inward. If `edgecorrect=TRUE` then the algorithm is modified to correct this vector bias.

Value

A point pattern (object of class "ppp") in the same window as the original pattern `X`, and with the same marks as `X`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

References

Choi, E. and Hall, P. (2001) Nonparametric analysis of earthquake point-process data. In M. de Gunst, C. Klaassen and A. van der Vaart (eds.) *State of the art in probability and statistics: Festschrift for Willem R. van Zwet*, Institute of Mathematical Statistics, Beachwood, Ohio. Pages 324–344.

See Also

density.ppp, Smooth.ppp.

Examples

```
data(shapley)
X <- unmark(shapley)

Y <- sharpen(X, sigma=0.5)
Z <- sharpen(X, sigma=0.5, edgecorrect=TRUE)
opa <- par(mar=rep(0.2, 4))
plot(solist(X, Y, Z), main= " ",
     main.panel=c("data", "sharpen", "sharpen, correct"),
     pch=".", equal.scales=TRUE, mar.panel=0.2)
par(opa)
```

---

shift *Apply Vector Translation*

Description

Applies a vector shift of the plane to a geometrical object, such as a point pattern or a window.

Usage

```
shift(X, ...)
```

Arguments

X	Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), or a window (object of class "owin").
...	Arguments determining the shift vector.

Details

This is generic. Methods are provided for point patterns (shift.ppp) and windows (shift.owin).

The object is translated by the vector vec.

Value

Another object of the same type, representing the result of applying the shift.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

shift.ppp, shift.owin, rotate, affine, periodify

---

shift.im *Apply Vector Translation To Pixel Image*

Description

Applies a vector shift to a pixel image

Usage

```
## S3 method for class 'im'
shift(X, vec=c(0,0), ..., origin=NULL)
```

Arguments

X	Pixel image (object of class "im").
vec	Vector of length 2 representing a translation.
...	Ignored
origin	Character string determining a location that will be shifted to the origin. Options are "centroid", "midpoint" and "bottomleft". Partially matched.

Details

The spatial location of each pixel in the image is translated by the vector vec. This is a method for the generic function shift.

If origin is given, then it should be one of the character strings "centroid", "midpoint" or "bottomleft". The argument vec will be ignored; instead the shift will be performed so that the specified geometric location is shifted to the origin. If origin="centroid" then the centroid of the image window will be shifted to the origin. If origin="midpoint" then the centre of the bounding rectangle of the image will be shifted to the origin. If origin="bottomleft" then the bottom left corner of the bounding rectangle of the image will be shifted to the origin.

Value

Another pixel image (of class "im") representing the result of applying the vector shift.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

shift

Examples

```
# make up an image
X <- setcov(unit.square())
plot(X)

Y <- shift(X, c(10,10))
plot(Y)
# no discernible difference except coordinates are different

shift(X, origin="mid")
```

`shift.owin` *Apply Vector Translation To Window*

Description

Applies a vector shift to a window

Usage

```
## S3 method for class 'owin'
shift(X, vec=c(0,0), ..., origin=NULL)
```

Arguments

`X`	Window (object of class `"owin"`).
`vec`	Vector of length 2 representing a translation.
`...`	Ignored
`origin`	Character string determining a location that will be shifted to the origin. Options are `"centroid"`, `"midpoint"` and `"bottomleft"`. Partially matched.

Details

The window is translated by the vector `vec`. This is a method for the generic function `shift`.

If `origin` is given, then it should be one of the character strings `"centroid"`, `"midpoint"` or `"bottomleft"`. The argument `vec` will be ignored; instead the shift will be performed so that the specified geometric location is shifted to the origin. If `origin="centroid"` then the centroid of the window will be shifted to the origin. If `origin="midpoint"` then the centre of the bounding rectangle of the window will be shifted to the origin. If `origin="bottomleft"` then the bottom left corner of the bounding rectangle of the window will be shifted to the origin.

Value

Another window (of class `"owin"`) representing the result of applying the vector shift.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

shift, shift.ppp, periodify, rotate, affine, centroid.owin

Examples

```
  W <- owin(c(0,1),c(0,1))
  X <- shift(W, c(2,3))
  ## Not run:
  plot(W)
  # no discernible difference except coordinates are different

## End(Not run)
  shift(W, origin="mid")
```

---

shift.ppp *Apply Vector Translation To Point Pattern*

Description

Applies a vector shift to a point pattern.

Usage

```
 ## S3 method for class 'ppp'
shift(X, vec=c(0,0), ..., origin=NULL)
```

Arguments

X	Point pattern (object of class "ppp").
vec	Vector of length 2 representing a translation.
...	Ignored
origin	Character string determining a location that will be shifted to the origin. Options are "centroid", "midpoint" and "bottomleft". Partially matched.

Details

The point pattern, and its window, are translated by the vector vec.

This is a method for the generic function shift.

If origin is given, then it should be one of the character strings "centroid", "midpoint" or "bottomleft". The argument vec will be ignored; instead the shift will be performed so that the specified geometric location is shifted to the origin. If origin="centroid" then the centroid of the window will be shifted to the origin. If origin="midpoint" then the centre of the bounding rectangle of the window will be shifted to the origin. If origin="bottomleft" then the bottom left corner of the bounding rectangle of the window will be shifted to the origin.

Value

Another point pattern (of class "ppp") representing the result of applying the vector shift.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

shift, shift.owin, periodify, rotate, affine

Examples

```
  data(cells)
  X <- shift(cells, c(2,3))
  ## Not run:
  plot(X)
  # no discernible difference except coordinates are different

## End(Not run)
  plot(cells, pch=16)
  plot(shift(cells, c(0.03,0.03)), add=TRUE)

  shift(cells, origin="mid")
```

---

shift.psp *Apply Vector Translation To Line Segment Pattern*

Description

Applies a vector shift to a line segment pattern.

Usage

```
 ## S3 method for class 'psp'
shift(X, vec=c(0,0), ..., origin=NULL)
```

Arguments

X	Line Segment pattern (object of class "psp").
vec	Vector of length 2 representing a translation.
...	Ignored
origin	Character string determining a location that will be shifted to the origin. Options are "centroid", "midpoint" and "bottomleft". Partially matched.

Details

The line segment pattern, and its window, are translated by the vector vec.

This is a method for the generic function shift.

If origin is given, then it should be one of the character strings "centroid", "midpoint" or "bottomleft". The argument vec will be ignored; instead the shift will be performed so that the specified geometric location is shifted to the origin. If origin="centroid" then the centroid of the window will be shifted to the origin. If origin="midpoint" then the centre of the bounding rectangle of the window will be shifted to the origin. If origin="bottomleft" then the bottom left corner of the bounding rectangle of the window will be shifted to the origin.

Value

Another line segment pattern (of class "psp") representing the result of applying the vector shift.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

shift, shift.owin, shift.ppp, periodify, rotate, affine

Examples

```
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(X, col="red")
Y <- shift(X, c(0.05,0.05))
plot(Y, add=TRUE, col="blue")

shift(Y, origin="mid")
```

---

sidelengths.owin *Side Lengths of Enclosing Rectangle of a Window*

Description

Computes the side lengths of the (enclosing rectangle of) a window.

Usage

```
## S3 method for class 'owin'
sidelengths(x)

## S3 method for class 'owin'
shortside(x)
```

Arguments

x	A window whose side lengths will be computed. Object of class "owin".

Details

The functions `shortside` and `sidelengths` are generic. The functions documented here are the methods for the class `"owin"`.

`sidelengths.owin` computes the side-lengths of the enclosing rectangle of the window `x`.

For safety, both functions give a warning if the window is not a rectangle. To suppress the warning, first convert the window to a rectangle using `as.rectangle`.

`shortside.owin` computes the minimum of the two side-lengths.

Value

For `sidelengths.owin`, a numeric vector of length 2 giving the side-lengths (x then y) of the enclosing rectangle. For `shortside.owin`, a numeric value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`shortside`, `sidelengths` for the generic functions.

`area.owin`, `diameter.owin`, `perimeter` for other geometric calculations on `"owin"` objects.

`owin`, `as.owin`.

Examples

```
w <- owin(c(0,2),c(-1,3))
sidelengths(w)
shortside(as.rectangle(letterR))
```

`simplepanel` *Simple Point-and-Click Interface Panels*

Description

These functions enable the user to create a simple, robust, point-and-click interface to any R code.

Usage

```
simplepanel(title, B, boxes, clicks,
   redraws=NULL, exit = NULL, env)

grow.simplepanel(P, side = c("right", "left", "top", "bottom"),
   len = NULL, new.clicks, new.redraws=NULL, ..., aspect)
```

Arguments

`title`	Character string giving the title of the interface panel.
`B`	Bounding box of the panel coordinates. A rectangular window (object of class `"owin"`)
`boxes`	A list of rectangular windows (objects of class `"owin"`) specifying the placement of the buttons and other interactive components of the panel.
`clicks`	A list of R functions, of the same length as `boxes`, specifying the operations to be performed when each button is clicked. Entries can also be `NULL` indicating that no action should occur. See Details.
`redraws`	Optional list of R functions, of the same length as `boxes`, specifying how to redraw each button. Entries can also be `NULL` indicating a simple default. See Details.
`exit`	An R function specifying actions to be taken when the interactive panel terminates.
`env`	An `environment` that will be passed as an argument to all the functions in `clicks`, `redraws` and `exit`.
`P`	An existing interaction panel (object of class `"simplepanel"`).
`side`	Character string identifying which side of the panel `P` should be grown to accommodate the new buttons.
`len`	Optional. Thickness of the new panel area that should be grown to accommodate the new buttons. A single number in the same units as the coordinate system of `P`.
`new.clicks`	List of R functions defining the operations to be performed when each of the new buttons is clicked.
`new.redraws`	Optional. List of R functions, of the same length as `new.clicks`, defining how to redraw each of the new buttons.
`...`	Arguments passed to `layout.boxes` to determine the layout of the new buttons.
`aspect`	Optional. Aspect ratio (height/width) of the new buttons.

Details

These functions enable the user to create a simple, robust, point-and-click interface to any R code.

The functions `simplepanel` and `grow.simplepanel` create an object of class `"simplepanel"`. Such an object defines the graphics to be displayed and the actions to be performed when the user interacts with the panel.

The panel is activated by calling `run.simplepanel`.

The function `simplepanel` creates a panel object from basic data. The function `grow.simplepanel` modifies an existing panel object `P` by growing an additional row or column of buttons.

For `simplepanel`,

- The spatial layout of the panel is determined by the rectangles `B` and `boxes`.
- The argument `clicks` must be a list of functions specifying the action to be taken when each button is clicked (or `NULL` to indicate that no action should be taken). The list entries should have names (but there are sensible defaults). Each function should be of the form `function(env, xy)` where `env` is an `environment` that may contain shared data, and `xy` gives the coordinates of the mouse click, in the format `list(x, y)`. The function returns `TRUE` if the panel should continue running, and `FALSE` if the panel should terminate.

- The argument `redraws`, if given, must be a list of functions specifying the action to be taken when each button is to be redrawn. Each function should be of the form `function(button, name, env)` where `button` is a rectangle specifying the location of the button in the current coordinate system; `name` is a character string giving the name of the button; and `env` is the `environment` that may contain shared data. The function returns `TRUE` if the panel should continue running, and `FALSE` if the panel should terminate. If `redraws` is not given (or if one of the entries in `redraws` is `NULL`), the default action is to draw a pink rectangle showing the button position, draw the name of the button in the middle of this rectangle, and return `TRUE`.
- The argument `exit`, if given, must be a function specifying the action to be taken when the panel terminates. (Termination occurs when one of the `clicks` functions returns `FALSE`). The `exit` function should be of the form `function(env)` where `env` is the `environment` that may contain shared data. Its return value will be used as the return value of `run.simplepanel`.
- The argument `env` should be an R environment. The panel buttons will have access to this environment, and will be able to read and write data in it. This mechanism is used to exchange data between the panel and other R code.

For `grow.simplepanel`,

- the spatial layout of the new boxes is determined by the arguments `side`, `len`, `aspect` and by the additional `...` arguments passed to `layout.boxes`.
- the argument `new.clicks` should have the same format as `clicks`. It implicitly specifies the number of new buttons to be added, and the actions to be performed when they are clicked.
- the optional argument `new.redraws`, if given, should have the same format as `redraws`. It specifies the actions to be performed when the new buttons are clicked.

Value

An object of class `"simplepanel"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`run.simplepanel`, `layout.boxes`

Examples

```
# make boxes (alternatively use layout.boxes())
Bminus <- square(1)
Bvalue <- shift(Bminus, c(1.2, 0))
Bplus <- shift(Bvalue, c(1.2, 0))
Bdone <- shift(Bplus, c(1.2, 0))
myboxes <- list(Bminus, Bvalue, Bplus, Bdone)
myB <- do.call(boundingbox,myboxes)

# make environment containing an integer count
myenv <- new.env()
assign("answer", 0, envir=myenv)

# what to do when finished: return the count.
myexit <- function(e) { return(get("answer", envir=e)) }
```

```
# button clicks
# decrement the count
Cminus <- function(e, xy) {
  ans <- get("answer", envir=e)
  assign("answer", ans - 1, envir=e)
  return(TRUE)
}
# display the count (clicking does nothing)
Cvalue <- function(...) { TRUE }
# increment the count
Cplus <- function(e, xy) {
  ans <- get("answer", envir=e)
  assign("answer", ans + 1, envir=e)
  return(TRUE)
}
# 'Clear' button
Cclear <- function(e, xy) {
  assign("answer", 0, envir=e)
  return(TRUE)
}
# quit button
Cdone <- function(e, xy) { return(FALSE) }

myclicks <- list("-"=Cminus,
                 value=Cvalue,
                 "+"=Cplus,
                 done=Cdone)

# redraw the button that displays the current value of the count
Rvalue <- function(button, nam, e) {
   plot(button, add=TRUE)
   ans <- get("answer", envir=e)
   text(centroid.owin(button), labels=ans)
   return(TRUE)
}

# make the panel
P <- simplepanel("Counter",
                 B=myB, boxes=myboxes,
                 clicks=myclicks,
                 redraws = list(NULL, Rvalue, NULL, NULL),
                 exit=myexit, env=myenv)
P
# ( type run.simplepanel(P) to run the panel interactively )

# add another button to right
Pplus <- grow.simplepanel(P, "right", new.clicks=list(clear=Cclear))
```

`simplify.owin` *Approximate a Polygon by a Simpler Polygon*

Description

Given a polygonal window, this function finds a simpler polygon that approximates it.

Usage

```
simplify.owin(W, dmin)
```

Arguments

W	The polygon which is to be simplied. An object of class "owin".
dmin	Numeric value. The smallest permissible length of an edge.

Details

This function simplifies a polygon W by recursively deleting the shortest edge of W until all remaining edges are longer than the specified minimum length dmin, or until there are only three edges left.

The argument W must be a window (object of class "owin"). It should be of type "polygonal". If W is a rectangle, it is returned without alteration.

The simplification algorithm is not yet implemented for binary masks. If W is a mask, an error is generated.

Value

Another window (object of class "owin") of type "polygonal".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

owin

Examples

```
plot(letterR, col="red")
plot(simplify.owin(letterR, 0.3), col="blue", add=TRUE)

W <- Window(chorley)
plot(W)
WS <- simplify.owin(W, 2)
plot(WS, add=TRUE, border="green")
points(vertices(WS))
```

---

simulate.dppm *Simulation of Determinantal Point Process Model*

Description

Generates simulated realisations from a determinantal point process model.

Usage

```
## S3 method for class 'dppm'
simulate(object, nsim = 1, seed = NULL, ...,
  W = NULL, trunc = 0.99, correction = "periodic", rbord = reach(object))

## S3 method for class 'detpointprocfamily'
simulate(object, nsim = 1, seed = NULL, ...,
  W = NULL, trunc = 0.99, correction = "periodic", rbord = reach(object))
```

Arguments

object	Determinantal point process model. An object of class "detpointprocfamily" or "dppm".
nsim	Number of simulated realisations.
seed	an object specifying whether and how to initialise the random number generator. Either NULL or an integer that will be used in a call to set.seed before simulating the point patterns.
...	Arguments passed on to rdpp.
W	Object specifying the window of simulation (defaults to a unit box if nothing else is sensible – see Details). Can be any single argument acceptable to as.boxx (e.g. an "owin", "box3" or "boxx" object).
trunc	Numeric value specifying how the model truncation is preformed. See Details.
correction	Character string specifying the type of correction to use. The options are "periodic" (default) and "border". See Details.
rbord	Numeric value specifying the extent of the border correction if this correction is used. See Details.

Details

These functions are methods for the generic function simulate for the classes "detpointprocfamily" and "dppm" of determinantal point process models.

The return value is a list of nsim point patterns. It also carries an attribute "seed" that captures the initial state of the random number generator. This follows the convention used in simulate.lm (see simulate). It can be used to force a sequence of simulations to be repeated exactly, as shown in the examples for simulate.

The exact simulation of a determinantal point process model involves an infinite series, which typically has no analytical solution. In the implementation a truncation is performed. The truncation trunc can be specified either directly as a positive integer or as a fraction between 0 and 1. In the latter case the truncation is chosen such that the expected number of points in a simulation is trunc times the theoretical expected number of points in the model. The default is 0.99.

The window of the returned point pattern(s) can be specified via the argument W. For a fitted model (of class "dppm") it defaults to the observation window of the data used to fit the model. For inhomogeneous models it defaults to the window of the intensity image. Otherwise it defaults to a unit box. For non-rectangular windows simulation is done in the containing rectangle and then restricted to the window. For inhomogeneous models a stationary model is first simulated using the maximum intensity and then the result is obtained by thinning.

The default is to use periodic edge correction for simulation such that opposite edges are glued together. If border correction is used then the simulation is done in an extended window. Edge effects are theoretically completely removed by doubling the size of the window in each spatial dimension, but for practical purposes much less extension may be sufficient. The numeric rbord determines the extent of the extra space added to the window.

Value

A list of length nsim containing simulated point patterns. If the patterns are two-dimensional, then they are objects of class "ppp", and the list has class "solist". Otherwise, the patterns are objects of class "ppx" and the list has class "anylist".

The return value also carries an attribute "seed" that captures the initial state of the random number generator. See Details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Lavancier, F. Møller, J. and Rubak, E. (2015) Determinantal point process models and statistical inference *Journal of the Royal Statistical Society, Series B* **77**, 853–977.

See Also

rdpp, simulate

Examples

```
model <- dppGauss(lambda=100, alpha=.05, d=2)
simulate(model, 2)
```

---

simulate.kppm — *Simulate a Fitted Cluster Point Process Model*

Description

Generates simulated realisations from a fitted cluster point process model.

Usage

```
  ## S3 method for class 'kppm'
simulate(object, nsim = 1, seed=NULL, ...,
         window=NULL, covariates=NULL, verbose=TRUE, retry=10,
         drop=FALSE)
```

Arguments

object	Fitted cluster point process model. An object of class "kppm".
nsim	Number of simulated realisations.
seed	an object specifying whether and how to initialise the random number generator. Either NULL or an integer that will be used in a call to set.seed before simulating the point patterns.
...	Ignored.
window	Optional. Window (object of class "owin") in which the model should be simulated.
covariates	Optional. A named list containing new values for the covariates in the model.
verbose	Logical. Whether to print progress reports (when nsim > 1).
retry	Number of times to repeat the simulation if it fails (e.g. because of insufficient memory).
drop	Logical. If nsim=1 and drop=TRUE, the result will be a point pattern, rather than a list containing a point pattern.

Details

This function is a method for the generic function simulate for the class "kppm" of fitted cluster point process models.

Simulations are performed by rThomas, rMatClust or rLGCP depending on the model.

The return value is a list of point patterns. It also carries an attribute "seed" that captures the initial state of the random number generator. This follows the convention used in simulate.lm (see simulate). It can be used to force a sequence of simulations to be repeated exactly, as shown in the examples for simulate.

Value

A list of length nsim containing simulated point patterns (objects of class "ppp").

The return value also carries an attribute "seed" that captures the initial state of the random number generator. See Details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

kppm, rThomas, rMatClust, rLGCP, simulate.ppm, simulate

Examples

```
data(redwood)
fit <- kppm(redwood, ~1, "Thomas")
simulate(fit, 2)
```

simulate.lppm *Simulate a Fitted Point Process Model on a Linear Network*

Description

Generates simulated realisations from a fitted Poisson point process model on a linear network.

Usage

```
  ## S3 method for class 'lppm'
simulate(object, nsim=1, ...,
                         new.coef=NULL,
                         progress=(nsim > 1),
                         drop=FALSE)
```

Arguments

`object`	Fitted point process model on a linear network. An object of class "lppm".
`nsim`	Number of simulated realisations.
`progress`	Logical flag indicating whether to print progress reports for the sequence of simulations.
`new.coef`	New values for the canonical parameters of the model. A numeric vector of the same length as `coef(object)`.
`...`	Arguments passed to `predict.lppm` to determine the spatial resolution of the image of the fitted intensity used in the simulation.
`drop`	Logical. If `nsim=1` and `drop=TRUE`, the result will be a point pattern, rather than a list containing a point pattern.

Details

This function is a method for the generic function `simulate` for the class "lppm" of fitted point process models on a linear network.

Only Poisson process models are supported so far.

Simulations are performed by `rpoislpp`.

Value

A list of length `nsim` containing simulated point patterns (objects of class "lpp") on the same linear network as the original data used to fit the model. The result also belongs to the class "solist", so that it can be plotted, and the class "timed", so that the total computation time is recorded.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`lppm`, `rpoislpp`, `simulate`

Examples

```
fit <- lppm(unmark(chicago) ~ y)
simulate(fit)[[1]]
```

---

simulate.mppm *Simulate a Point Process Model Fitted to Several Point Patterns*

Description

Generates simulated realisations from a point process model that was fitted to several point patterns.

Usage

```
## S3 method for class 'mppm'
simulate(object, nsim=1, ..., verbose=TRUE)
```

Arguments

object	Point process model fitted to several point patterns. An object of class "mppm".
nsim	Number of simulated realisations (of each original pattern).
...	Further arguments passed to simulate.ppm to control the simulation.
verbose	Logical value indicating whether to print progress reports.

Details

This function is a method for the generic function simulate for the class "mppm" of fitted point process models for replicated point pattern data.

The result is a hyperframe with n rows and nsim columns, where n is the number of original point pattern datasets to which the model was fitted. Each column of the hyperframe contains a simulated version of the original data.

For each of the original point pattern datasets, the fitted model for this dataset is extracted using subfits, then nsim simulated realisations of this model are generated using simulate.ppm, and these are stored in the corresponding row of the output.

Value

A hyperframe.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

mppm, simulate.ppm.

Examples

```
H <- hyperframe(Bugs=waterstriders)
fit <- mppm(Bugs ~ id, H)
y <- simulate(fit, nsim=2)
y
plot(y[1,,drop=TRUE], main="Simulations for Waterstriders pattern 1")
plot(y[,1,drop=TRUE], main="Simulation 1 for each Waterstriders pattern")
```

---

simulate.ppm *Simulate a Fitted Gibbs Point Process Model*

Description

Generates simulated realisations from a fitted Gibbs or Poisson point process model.

Usage

```
## S3 method for class 'ppm'
simulate(object, nsim=1, ...,
                          singlerun = FALSE,
                          start = NULL,
                          control = default.rmhcontrol(object, w=w),
                          w = NULL,
                          project=TRUE, new.coef=NULL,
                          verbose=FALSE, progress=(nsim > 1),
                          drop=FALSE)
```

Arguments

object	Fitted point process model. An object of class "ppm".
nsim	Number of simulated realisations.
singlerun	Logical. Whether to generate the simulated realisations from a single long run of the Metropolis-Hastings algorithm (singlerun=TRUE) or from separate, independent runs of the algorithm (singlerun=FALSE, the default).
start	Data determining the initial state of the Metropolis-Hastings algorithm. See rmhstart for description of these arguments. Defaults to list(n.start=npoints(data.ppm(objec meaning that the initial state of the algorithm has the same number of points as the original dataset.
control	Data controlling the running of the Metropolis-Hastings algorithm. See rmhcontrol for description of these arguments.
w	Optional. The window in which the model is defined. An object of class "owin".
...	Further arguments passed to rmhcontrol, or to rmh.default, or to covariate functions in the model.
project	Logical flag indicating what to do if the fitted model is invalid (in the sense that the values of the fitted coefficients do not specify a valid point process). If project=TRUE the closest valid model will be simulated; if project=FALSE an error will occur.
verbose	Logical flag indicating whether to print progress reports from rmh.ppm during the simulation of each point pattern.

progress	Logical flag indicating whether to print progress reports for the sequence of simulations.
new.coef	New values for the canonical parameters of the model. A numeric vector of the same length as coef(object).
drop	Logical. If nsim=1 and drop=TRUE, the result will be a point pattern, rather than a list containing a point pattern.

Details

This function is a method for the generic function simulate for the class "ppm" of fitted point process models.

Simulations are performed by rmh.ppm.

If singlerun=FALSE (the default), the simulated patterns are the results of independent runs of the Metropolis-Hastings algorithm. If singlerun=TRUE, a single long run of the algorithm is performed, and the state of the simulation is saved every nsave iterations to yield the simulated patterns.

In the case of a single run, the behaviour is controlled by the parameters nsave,nburn,nrep. These are described in rmhcontrol. They may be passed in the ... arguments or included in control. It is sufficient to specify two of the three parameters nsave,nburn,nrep.

Value

A list of length nsim containing simulated point patterns (objects of class "ppp"). It also belongs to the class "solist", so that it can be plotted, and the class "timed", so that the total computation time is recorded.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, simulate.kppm, simulate

Examples

```
fit <- ppm(japanesepines, ~1, Strauss(0.1))
simulate(fit, 2)
simulate(fit, 2, singlerun=TRUE, nsave=1e4, nburn=1e4)
```

simulate.slrm	*Simulate a Fitted Spatial Logistic Regression Model*

Description

Generates simulated realisations from a fitted spatial logistic regresson model

Usage

```
  ## S3 method for class 'slrm'
simulate(object, nsim = 1, seed=NULL, ...,
         window=NULL, covariates=NULL, verbose=TRUE, drop=FALSE)
```

Arguments

`object`	Fitted spatial logistic regression model. An object of class "`slrm`".
`nsim`	Number of simulated realisations.
`seed`	an object specifying whether and how to initialise the random number generator. Either `NULL` or an integer that will be used in a call to `set.seed` before simulating the point patterns.
`...`	Ignored.
`window`	Optional. Window (object of class "`owin`") in which the model should be simulated.
`covariates`	Optional. A named list containing new values for the covariates in the model.
`verbose`	Logical. Whether to print progress reports (when `nsim > 1`).
`drop`	Logical. If `nsim=1` and `drop=TRUE`, the result will be a point pattern, rather than a list containing a point pattern.

Details

This function is a method for the generic function `simulate` for the class "`slrm`" of fitted spatial logistic regression models.

Simulations are performed by `rpoispp` after the intensity has been computed by `predict.slrm`.

The return value is a list of point patterns. It also carries an attribute "`seed`" that captures the initial state of the random number generator. This follows the convention used in `simulate.lm` (see `simulate`). It can be used to force a sequence of simulations to be repeated exactly, as shown in the examples for `simulate`.

Value

A list of length `nsim` containing simulated point patterns (objects of class "`ppp`").

The return value also carries an attribute "`seed`" that captures the initial state of the random number generator. See Details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

slrm, rpoispp, simulate.ppm, simulate.kppm, simulate

Examples

```
X <- copper$SouthPoints
fit <- slrm(X ~ 1)
simulate(fit, 2)
fitxy <- slrm(X ~ x+y)
simulate(fitxy, 2, window=square(2))
```

---

slrm *Spatial Logistic Regression*

Description

Fits a spatial logistic regression model to a spatial point pattern.

Usage

```
slrm(formula, ..., data = NULL, offset = TRUE, link = "logit",
                     dataAtPoints=NULL, splitby=NULL)
```

Arguments

formula	The model formula. See Details.
...	Optional arguments passed to pixellate determining the pixel resolution for the discretisation of the point pattern.
data	Optional. A list containing data required in the formula. The names of entries in the list should correspond to variable names in the formula. The entries should be point patterns, pixel images or windows.
offset	Logical flag indicating whether the model formula should be augmented by an offset equal to the logarithm of the pixel area.
link	The link function for the regression model. A character string, specifying a link function for binary regression.
dataAtPoints	Optional. Exact values of the covariates at the data points. A data frame, with column names corresponding to variables in the formula, with one row for each point in the point pattern dataset.
splitby	Optional. Character string identifying a window. The window will be used to split pixels into sub-pixels.

Details

This function fits a Spatial Logistic Regression model (Tukey, 1972; Agterberg, 1974) to a spatial point pattern dataset. The logistic function may be replaced by another link function.

The formula specifies the form of the model to be fitted, and the data to which it should be fitted. The formula must be an R formula with a left and right hand side.

The left hand side of the formula is the name of the point pattern dataset, an object of class "ppp".

The right hand side of the formula is an expression, in the usual R formula syntax, representing the functional form of the linear predictor for the model.

Each variable name that appears in the formula may be

- one of the reserved names x and y, referring to the Cartesian coordinates;
- the name of an entry in the list data, if this argument is given;
- the name of an object in the parent environment, that is, in the environment where the call to slrm was issued.

Each object appearing on the right hand side of the formula may be

- a pixel image (object of class "im") containing the values of a covariate;
- a window (object of class "owin"), which will be interpreted as a logical covariate which is TRUE inside the window and FALSE outside it;
- a function in the R language, with arguments x,y, which can be evaluated at any location to obtain the values of a covariate.

See the Examples below.

The fitting algorithm discretises the point pattern onto a pixel grid. The value in each pixel is 1 if there are any points of the point pattern in the pixel, and 0 if there are no points in the pixel. The dimensions of the pixel grid will be determined as follows:

- The pixel grid will be determined by the extra arguments ... if they are specified (for example the argument dimyx can be used to specify the number of pixels).
- Otherwise, if the right hand side of the formula includes the names of any pixel images containing covariate values, these images will determine the pixel grid for the discretisation. The covariate image with the finest grid (the smallest pixels) will be used.
- Otherwise, the default pixel grid size is given by spatstat.options("npixel").

If link="logit" (the default), the algorithm fits a Spatial Logistic Regression model. This model states that the probability p that a given pixel contains a data point, is related to the covariates through

$$\log \frac{p}{1-p} = \eta$$

where η is the linear predictor of the model (a linear combination of the covariates, whose form is specified by the formula).

If link="cloglog" then the algorithm fits a model stating that

$$\log(-\log(1-p)) = \eta$$

.

If offset=TRUE (the default), the model formula will be augmented by adding an offset term equal to the logarithm of the pixel area. This ensures that the fitted parameters are approximately independent of pixel size. If offset=FALSE, the offset is not included, and the traditional form of Spatial Logistic Regression is fitted.

Value

An object of class "slrm" representing the fitted model.

There are many methods for this class, including methods for print, fitted, predict, anova, coef, logLik, terms, update, formula and vcov. Automated stepwise model selection is possible using step. Confidence intervals for the parameters can be computed using confint.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Agterberg, F.P. (1974) Automatic contouring of geological maps to detect target areas for mineral exploration. *Journal of the International Association for Mathematical Geology* **6**, 373–395.

Baddeley, A., Berman, M., Fisher, N.I., Hardegen, A., Milne, R.K., Schuhmacher, D., Shah, R. and Turner, R. (2010) Spatial logistic regression and change-of-support for spatial Poisson point processes. *Electronic Journal of Statistics* **4**, 1151–1201. doi: 10.1214/10-EJS581

Tukey, J.W. (1972) Discussion of paper by F.P. Agterberg and S.C. Robinson. *Bulletin of the International Statistical Institute* **44** (1) p. 596. Proceedings, 38th Congress, International Statistical Institute.

See Also

anova.slrm, coef.slrm, fitted.slrm, logLik.slrm, plot.slrm, predict.slrm, vcov.slrm

Examples

```
X <- copper$SouthPoints
slrm(X ~ 1)
slrm(X ~ x+y)

slrm(X ~ x+y, link="cloglog")
# specify a grid of 2-km-square pixels
slrm(X ~ 1, eps=2)

Y <- copper$SouthLines
Z <- distmap(Y)
slrm(X ~ Z)
slrm(X ~ Z, dataAtPoints=list(Z=nncross(X,Y,what="dist")))

dat <- list(A=X, V=Z)
slrm(A ~ V, data=dat)
```

Smooth — *Spatial smoothing of data*

Description

Generic function to perform spatial smoothing of spatial data.

Usage

```
Smooth(X, ...)
```

Arguments

X	Some kind of spatial data
...	Arguments passed to methods.

Details

This generic function calls an appropriate method to perform spatial smoothing on the spatial dataset X.

Methods for this function include

- Smooth.ppp for point patterns
- Smooth.msr for measures
- Smooth.fv for function value tables

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Smooth.ppp, Smooth.im, Smooth.msr, Smooth.fv.

Smooth.fv *Apply Smoothing to Function Values*

Description

Applies smoothing to the values in selected columns of a function value table.

Usage

```
## S3 method for class 'fv'
Smooth(X, which = "*", ...,
          method=c("smooth.spline", "loess"),
          xinterval=NULL)
```

Arguments

X	Values to be smoothed. A function value table (object of class "fv", see fv.object).
which	Character vector identifying which columns of the table should be smoothed. Either a vector containing names of columns, or one of the wildcard strings "*" or "." explained below.
...	Extra arguments passed to smooth.spline or loess to control the smoothing.
method	Smoothing algorithm. A character string, partially matched to either "smooth.spline" or "loess".
xinterval	Optional. Numeric vector of length 2 specifying a range of x values. Smoothing will be performed only on the part of the function corresponding to this range.

Details

The command `Smooth.fv` applies smoothing to the function values in a function value table (object of class `"fv"`).

`Smooth.fv` is a method for the generic function `Smooth`.

The smoothing is performed either by `smooth.spline` or by `loess`.

Smoothing is applied to every column (or to each of the selected columns) of function values in turn, using the function argument as the x coordinate and the selected column as the y coordinate. The original function values are then replaced by the corresponding smooth interpolated function values.

The optional argument `which` specifies which of the columns of function values in `x` will be smoothed. The default (indicated by the wildcard `which="*"`) is to smooth all function values, i.e.\ all columns except the function argument. Alternatively `which="."` designates the subset of function values that are displayed in the default plot. Alternatively `which` can be a character vector containing the names of columns of `x`.

If the argument `xinterval` is given, then smoothing will be performed only in the specified range of x values.

Value

Another function value table (object of class `"fv"`) of the same format.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`Smooth`, `with.fv`, `fv.object`, `smooth.spline`, `smooth.spline`

Examples

```
data(cells)
G <- Gest(cells)
plot(G)
plot(Smooth(G, df=9), add=TRUE)
```

`Smooth.msr` *Smooth a Signed or Vector-Valued Measure*

Description

Apply kernel smoothing to a signed measure or vector-valued measure.

Usage

```
 ## S3 method for class 'msr'
Smooth(X, ..., drop=TRUE)
```

Arguments

X	Object of class "msr" representing a signed measure or vector-valued measure.
...	Arguments passed to density.ppp controlling the smoothing bandwidth and the pixel resolution.
drop	Logical. If TRUE (the default), the result of smoothing a scalar-valued measure is a pixel image. If FALSE, the result of smoothing a scalar-valued measure is a list containing one pixel image.

Details

This function applies kernel smoothing to a signed measure or vector-valued measure X. The Gaussian kernel is used.

The object X would typically have been created by residuals.ppm or msr.

Value

A pixel image or a list of pixel images. For scalar-valued measures, a pixel image (object of class "im") provided drop=TRUE. For vector-valued measures (or if drop=FALSE), a list of pixel images; the list also belongs to the class "solist" so that it can be printed and plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

Baddeley, A., Møller, J. and Pakes, A.G. (2008) Properties of residuals for spatial point processes. *Annals of the Institute of Statistical Mathematics* **60**, 627–649.

See Also

Smooth, msr, plot.msr

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
rp <- residuals(fit, type="pearson")
rs <- residuals(fit, type="score")

plot(Smooth(rp))
plot(Smooth(rs))
```

Smooth.ppp *Spatial smoothing of observations at irregular points*

Description

Performs spatial smoothing of numeric values observed at a set of irregular locations. Uses Gaussian kernel smoothing and least-squares cross-validated bandwidth selection.

Usage

```
## S3 method for class 'ppp'
Smooth(X, sigma=NULL,
                     ...,
                     weights = rep(1, npoints(X)),
                     at="pixels",
                     edge=TRUE, diggle=FALSE, geometric=FALSE)

markmean(X, ...)

markvar(X, sigma=NULL, ..., weights=NULL, varcov=NULL)
```

Arguments

`X`	A marked point pattern (object of class "ppp").
`sigma`	Smoothing bandwidth. A single positive number, a numeric vector of length 2, or a function that selects the bandwidth automatically. See `density.ppp`.
`...`	Further arguments passed to `bw.smoothppp` and `density.ppp` to control the kernel smoothing and the pixel resolution of the result.
`weights`	Optional weights attached to the observations. A numeric vector, numeric matrix, an `expression` or a pixel image. See `density.ppp`.
`at`	String specifying whether to compute the smoothed values at a grid of pixel locations (`at="pixels"`) or only at the points of `X` (`at="points"`).
`edge,diggle`	Arguments passed to `density.ppp` to determine the edge correction.
`varcov`	Variance-covariance matrix. An alternative to `sigma`. See `density.ppp`.
`geometric`	Logical value indicating whether to perform geometric mean smoothing instead of arithmetic mean smoothing. See Details.

Details

The function `Smooth.ppp` performs spatial smoothing of numeric values observed at a set of irregular locations. The functions `markmean` and `markvar` are wrappers for `Smooth.ppp` which compute the spatially-varying mean and variance of the marks of a point pattern.

`Smooth.ppp` is a method for the generic function `Smooth` for the class "ppp" of point patterns. Thus you can type simply `Smooth(X)`.

Smoothing is performed by Gaussian kernel weighting. If the observed values are $v_1, \ldots, v_n$ at locations $x_1, \ldots, x_n$ respectively, then the smoothed value at a location u is (ignoring edge corrections)

$$g(u) = \frac{\sum_i k(u - x_i) v_i}{\sum_i k(u - x_i)}$$

where k is a Gaussian kernel. This is known as the Nadaraya-Watson smoother (Nadaraya, 1964, 1989; Watson, 1964). By default, the smoothing kernel bandwidth is chosen by least squares cross-validation (see below).

The argument X must be a marked point pattern (object of class "ppp", see ppp.object). The points of the pattern are taken to be the observation locations x_i, and the marks of the pattern are taken to be the numeric values v_i observed at these locations.

The marks are allowed to be a data frame (in Smooth.ppp and markmean). Then the smoothing procedure is applied to each column of marks.

The numerator and denominator are computed by density.ppp. The arguments ... control the smoothing kernel parameters and determine whether edge correction is applied. The smoothing kernel bandwidth can be specified by either of the arguments sigma or varcov which are passed to density.ppp. If neither of these arguments is present, then by default the bandwidth is selected by least squares cross-validation, using bw.smoothppp.

The optional argument weights allows numerical weights to be applied to the data. If a weight w_i is associated with location x_i, then the smoothed function is (ignoring edge corrections)

$$g(u) = \frac{\sum_i k(u - x_i) v_i w_i}{\sum_i k(u - x_i) w_i}$$

If geometric=TRUE then geometric mean smoothing is performed instead of arithmetic mean smoothing. The mark values must be non-negative numbers. The logarithm of the mark values is computed; these logarithmic values are kernel-smoothed as described above; then the exponential function is applied to the smoothed values.

An alternative to kernel smoothing is inverse-distance weighting, which is performed by idw.

Value

If X *has a single column of marks:*

- If at="pixels" (the default), the result is a pixel image (object of class "im"). Pixel values are values of the interpolated function.
- If at="points", the result is a numeric vector of length equal to the number of points in X. Entries are values of the interpolated function at the points of X.

If X *has a data frame of marks:*

- If at="pixels" (the default), the result is a named list of pixel images (object of class "im"). There is one image for each column of marks. This list also belongs to the class "solist", for which there is a plot method.
- If at="points", the result is a data frame with one row for each point of X, and one column for each column of marks. Entries are values of the interpolated function at the points of X.

The return value has attributes "sigma" and "varcov" which report the smoothing bandwidth that was used.

Very small bandwidth

If the chosen bandwidth sigma is very small, kernel smoothing is mathematically equivalent to nearest-neighbour interpolation; the result will be computed by nnmark. This is unless at="points" and leaveoneout=FALSE, when the original mark values are returned.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Nadaraya, E.A. (1964) On estimating regression. *Theory of Probability and its Applications* **9**, 141–142.

Nadaraya, E.A. (1989) *Nonparametric estimation of probability densities and regression curves*. Kluwer, Dordrecht.

Watson, G.S. (1964) Smooth regression analysis. *Sankhya A* **26**, 359–372.

See Also

Smooth,

density.ppp, bw.smoothppp, nnmark, ppp.object, im.object.

See idw for inverse-distance weighted smoothing.

To perform interpolation, see also the akima package.

Examples

```
# Longleaf data - tree locations, marked by tree diameter
# Local smoothing of tree diameter (automatic bandwidth selection)
Z <- Smooth(longleaf)
# Kernel bandwidth sigma=5
plot(Smooth(longleaf, 5))
# mark variance
plot(markvar(longleaf, sigma=5))
# data frame of marks: trees marked by diameter and height
plot(Smooth(finpines, sigma=2))
head(Smooth(finpines, sigma=2, at="points"))
```

Smooth.ssf *Smooth a Spatially Sampled Function*

Description

Applies kernel smoothing to a spatially sampled function.

Usage

```
## S3 method for class 'ssf'
Smooth(X, ...)
```

Arguments

X	Object of class "ssf".
...	Arguments passed to Smooth.ppp to control the smoothing.

Details

An object of class "ssf" represents a real-valued or vector-valued function that has been evaluated or sampled at an irregular set of points.

The function values will be smoothed using a Gaussian kernel.

Value

A pixel image or a list of pixel images.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

ssf, Smooth.ppp

Examples

```
f <- ssf(redwood, nndist(redwood))
Smooth(f, sigma=0.1)
```

Smoothfun.ppp *Smooth Interpolation of Marks as a Spatial Function*

Description

Perform spatial smoothing of numeric values observed at a set of irregular locations, and return the result as a function of spatial location.

Usage

```
Smoothfun(X, ...)

## S3 method for class 'ppp'
Smoothfun(X, sigma = NULL, ...,
                        weights = NULL, edge = TRUE, diggle = FALSE)
```

Arguments

X	Marked point pattern (object of class "ppp").
sigma	Smoothing bandwidth, or bandwidth selection function, passed to Smooth.ppp.
...	Additional arguments passed to Smooth.ppp.
weights	Optional vector of weights associated with the points of X.
edge,diggle	Logical arguments controlling the edge correction. Arguments passed to Smooth.ppp.

Details

The commands Smoothfun and Smooth both perform kernel-smoothed spatial interpolation of numeric values observed at irregular spatial locations. The difference is that Smooth returns a pixel image, containing the interpolated values at a grid of locations, while Smoothfun returns a function(x,y) which can be used to compute the interpolated value at *any* spatial location. For purposes such as model-fitting it is more accurate to use Smoothfun to interpolate data.

Value

A function with arguments x,y. The function also belongs to the class "Smoothfun" which has methods for print and as.im. It also belongs to the class "funxy" which has methods for plot, contour and persp.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

See Also

Smooth

Examples

```
f <- Smoothfun(longleaf)
f
f(120, 80)
plot(f)
```

Softcore *The Soft Core Point Process Model*

Description

Creates an instance of the Soft Core point process model which can then be fitted to point pattern data.

Usage

```
Softcore(kappa, sigma0=NA)
```

Arguments

kappa	The exponent κ of the Soft Core interaction
sigma0	Optional. Initial estimate of the parameter σ. A positive number.

Details

The (stationary) Soft Core point process with parameters β and σ and exponent κ is the pairwise interaction point process in which each point contributes a factor β to the probability density of the point pattern, and each pair of points contributes a factor

$$\exp\left\{-\left(\frac{\sigma}{d}\right)^{2/\kappa}\right\}$$

to the density, where d is the distance between the two points.

Thus the process has probability density

$$f(x_1,\ldots,x_n) = \alpha\beta^{n(x)} \exp\left\{-\sum_{i<j}\left(\frac{\sigma}{||x_i - x_j||}\right)^{2/\kappa}\right\}$$

where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, α is the normalising constant, and the sum on the right hand side is over all unordered pairs of points of the pattern.

This model describes an "ordered" or "inhibitive" process, with the interpoint interaction decreasing smoothly with distance. The strength of interaction is controlled by the parameter σ, a positive real number, with larger values corresponding to stronger interaction; and by the exponent κ in the range $(0, 1)$, with larger values corresponding to weaker interaction. If $\sigma = 0$ the model reduces to the Poisson point process. If $\sigma > 0$, the process is well-defined only for κ in $(0, 1)$. The limit of the model as $\kappa \to 0$ is the hard core process with hard core distance $h = \sigma$.

The nonstationary Soft Core process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location, rather than a constant beta.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Soft Core process pairwise interaction is yielded by the function `Softcore()`. See the examples below.

The main argument is the exponent `kappa`. When `kappa` is fixed, the model becomes an exponential family with canonical parameters $\log\beta$ and

$$\log\gamma = \frac{2}{\kappa}\log\sigma$$

The canonical parameters are estimated by `ppm()`, not fixed in `Softcore()`.

The optional argument `sigma0` can be used to improve numerical stability. If `sigma0` is given, it should be a positive number, and it should be a rough estimate of the parameter σ.

Value

An object of class `"interact"` describing the interpoint interaction structure of the Soft Core process with exponent κ.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Ogata, Y, and Tanemura, M. (1981). Estimation of interaction potentials of spatial point patterns through the maximum likelihood procedure. *Annals of the Institute of Statistical Mathematics*, B **33**, 315–338.

Ogata, Y, and Tanemura, M. (1984). Likelihood analysis of spatial point patterns. *Journal of the Royal Statistical Society, series B* **46**, 496–518.

See Also

ppm, pairwise.family, ppm.object

Examples

```
data(cells)
ppm(cells, ~1, Softcore(kappa=0.5), correction="isotropic")
# fit the stationary Soft Core process to `cells'
```

---

solapply — *Apply a Function Over a List and Obtain a List of Objects*

Description

Applies the function FUN to each element of the list X, and returns the result as a list of class "solist" or "anylist" as appropriate.

Usage

```
anylapply(X, FUN, ...)

solapply(X, FUN, ..., check = TRUE, promote = TRUE, demote = FALSE)
```

Arguments

X	A list.
FUN	Function to be applied to each element of X.
...	Additional arguments to FUN.
check,promote,demote	Arguments passed to solist which determine how to handle different classes of objects.

Details

These convenience functions are similar to lapply except that they return a list of class "solist" or "anylist".

In both functions, the result is computed by lapply(X, FUN, ...).

In anylapply the result is converted to a list of class "anylist" and returned.

In solapply the result is converted to a list of class "solist" **if possible**, using as.solist. If this is not possible, then the behaviour depends on the argument demote. If demote=TRUE the result will be returned as a list of class "anylist". If demote=FALSE (the default), an error occurs.

Value

A list, usually of class "solist".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

solist, anylist.

Examples

```
solapply(waterstriders, density)
```

---

solist *List of Two-Dimensional Spatial Objects*

Description

Make a list of two-dimensional spatial objects.

Usage

```
solist(..., check=TRUE, promote=TRUE, demote=FALSE)
```

Arguments

...	Any number of objects, each representing a two-dimensional spatial dataset.
check	Logical value. If TRUE, check that each of the objects is a 2D spatial object.
promote	Logical value. If TRUE, test whether all objects belong to the *same* class, and if so, promote the list of objects to the appropriate class of list.
demote	Logical value determining what should happen if any of the objects is not a 2D spatial object: if demote=FALSE (the default), a fatal error occurs; if demote=TRUE, a list of class "anylist" is returned.

Details

This command creates an object of class "solist" (spatial object list) which represents a list of two-dimensional spatial datasets. The datasets do not necessarily belong to the same class.

Typically the intention is that the datasets in the list should be treated in the same way, for example, they should be plotted side-by-side. The **spatstat** package provides a plotting function, plot.solist, and many other functions for this class.

In the **spatstat** package, various functions produce an object of class "solist". For example, when a point pattern is split into several point patterns by split.ppp, or an image is split into several images by split.im, the result is of class "solist".

If check=TRUE then the code will check whether all objects in ... belong to the classes of two-dimensional spatial objects defined in the **spatstat** package. They do not have to belong to the *same* class. Set check=FALSE for efficiency, but only if you are sure that all the objects are valid.

If some of the objects in ... are not two-dimensional spatial objects, the action taken depends on the argument demote. If demote=TRUE, the result will belong to the more general class "anylist" instead of "solist". If demote=FALSE (the default), an error occurs.

If promote=TRUE then the code will check whether all the objects ... belong to the same class. If they are all point patterns (class "ppp"), the result will also belong to the class "ppplist". If they are all pixel images (class "im"), the result will also belong to the class "imlist".

Use as.solist to convert a list to a "solist".

Value

A list, usually belonging to the class "solist".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

as.solist, anylist, solapply

Examples

```
solist(cells, density(cells))
solist(cells, japanesepines, redwood)
```

---

solutionset *Evaluate Logical Expression Involving Pixel Images and Return Region Where Expression is True*

Description

Given a logical expression involving one or more pixel images, find all pixels where the expression is true, and assemble these pixels into a window.

Usage

```
solutionset(..., envir)
```

Arguments

...	An expression in the R language, involving one or more pixel images.
envir	Optional. The environment in which to evaluate the expression.

Details

Given a logical expression involving one or more pixel images, this function will find all pixels where the expression is true, and assemble these pixels into a spatial window.

Pixel images in spatstat are represented by objects of class "im" (see im.object). These are essentially matrices of pixel values, with extra attributes recording the pixel dimensions, etc.

Suppose X is a pixel image. Then solutionset(abs(X) > 3) will find all the pixels in X for which the pixel value is greater than 3 in absolute value, and return a window containing all these pixels.

If X and Y are two pixel images, solutionset(X > Y) will find all pixels for which the pixel value of X is greater than the corresponding pixel value of Y, and return a window containing these pixels.

In general, ... can be any logical expression involving pixel images.

The code first tries to evaluate the expression using eval.im. This is successful if the expression involves only (a) the *names* of pixel images, (b) scalar constants, and (c) functions which are vectorised. There must be at least one pixel image in the expression. The expression expr must be vectorised. See the Examples.

If this is unsuccessful, the code then tries to evaluate the expression using pixel arithmetic. This is successful if all the arithmetic operations in the expression are listed in Math.im.

Value

A spatial window (object of class "owin", see owin.object).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

im.object, owin.object, eval.im, levelset

Examples

```
# test images
X <- as.im(function(x,y) { x^2 - y^2 }, unit.square())
Y <- as.im(function(x,y) { 3 * x + y  - 1}, unit.square())

W <- solutionset(abs(X) > 0.1)
W <- solutionset(X > Y)
W <- solutionset(X + Y >= 1)

area(solutionset(X < Y))

solutionset(density(cells) > 20)
```

spatdim *Spatial Dimension of a Dataset*

Description

Extracts the spatial dimension of an object in the **spatstat** package.

Usage

```
spatdim(X)
```

Arguments

X	Object belonging to any class defined in the **spatstat** package.

Details

This function returns the number of spatial coordinate dimensions of the dataset X. The results for some of the more common types of objects are as follows:

object class	**dimension**
"ppp"	2
"lpp"	2
"pp3"	3
"ppx"	number of *spatial* dimensions
"owin"	2
"psp"	2
"ppm"	2

Note that time dimensions are not counted.

If X is not a recognised spatial object, the result is NA.

Value

An integer, or NA.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
spatdim(lansing)
```

---

spatialcdf *Spatial Cumulative Distribution Function*

Description

Compute the spatial cumulative distribution function of a spatial covariate, optionally using spatially-varying weights.

Usage

```
spatialcdf(Z, weights = NULL, normalise = FALSE, ..., W = NULL, Zname = NULL)
```

Arguments

Z	Spatial covariate. A pixel image or a `function(x,y,...)`
weights	Spatial weighting for different locations. A pixel image, a `function(x,y,...)`, a window, a constant value, or a fitted point process model (object of class `"ppm"` or `"kppm"`).
normalise	Logical. Whether the weights should be normalised so that they sum to 1.
...	Arguments passed to `as.mask` to determine the pixel resolution, or extra arguments passed to `Z` if it is a function.
W	Optional window (object of class `"owin"`) defining the spatial domain.
Zname	Optional character string for the name of the covariate `Z` used in plots.

Details

If `weights` is missing or `NULL`, it defaults to 1. The values of the covariate `Z` are computed on a grid of pixels. The weighted cumulative distribution function of `Z` values is computed, taking each value with weight equal to the pixel area. The resulting function F is such that $F(t)$ is the area of the region of space where $Z \leq t$.

If `weights` is a pixel image or a function, then the values of `weights` and of the covariate `Z` are computed on a grid of pixels. The `weights` are multiplied by the pixel area. Then the weighted empirical cumulative distribution function of `Z` values is computed using `ewcdf`. The resulting function F is such that $F(t)$ is the total weight (or weighted area) of the region of space where $Z \leq t$.

If `weights` is a fitted point process model, then it should be a Poisson process. The fitted intensity of the model, and the value of the covariate `Z`, are evaluated at the quadrature points used to fit the model. The `weights` are multiplied by the weights of the quadrature points. Then the weighted empirical cumulative distribution of `Z` values is computed using `ewcdf`. The resulting function F is such that $F(t)$ is the expected number of points in the point process that will fall in the region of space where $Z \leq t$.

If `normalise=TRUE`, the function is normalised so that its maximum value equals 1, so that it gives the cumulative *fraction* of weight or cumulative fraction of points.

The result can be printed, plotted, and used as a function.

Value

A cumulative distribution function object belonging to the classes `"spatialcdf"`, `"ewcdf"`, `"ecdf"` and `"stepfun"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

ewcdf, cdf.test

Examples

```
 with(bei.extra, {
   plot(spatialcdf(grad))
   fit <- ppm(bei ~ elev)
   plot(spatialcdf(grad, predict(fit)))
   plot(A <- spatialcdf(grad, fit))
   A(0.1)
})
```

---

spatstat.options *Internal Options in Spatstat Package*

Description

Allows the user to examine and reset the values of global parameters which control actions in the **spatstat** package.

Usage

```
spatstat.options(...)
reset.spatstat.options()
```

Arguments

...	Either empty, or a succession of parameter names in quotes, or a succession of name=value pairs. See below for the parameter names.

Details

The function spatstat.options allows the user to examine and reset the values of global parameters which control actions in the **spatstat** package. It is analogous to the system function options.

The function reset.spatstat.options resets all the global parameters in **spatstat** to their original, default values.

The global parameters of interest to the user are:

checkpolygons Logical flag indicating whether the functions owin and as.owin should apply very strict checks on the validity of polygon data. These strict checks are no longer necessary, and the default is checkpolygons=FALSE. See also fixpolygons below.

checksegments Logical flag indicating whether the functions psp and as.psp should check the validity of line segment data (in particular, checking that the endpoints of the line segments are inside the specified window). It is advisable to leave this flag set to TRUE.

eroded.intensity Logical flag affecting the behaviour of the score and pseudo-score residual functions Gcom, Gres Kcom, Kres, psstA, psstG, psst. The flag indicates whether to compute intensity estimates on an eroded window (eroded.intensity=TRUE) or on the original data window (eroded.intensity=FALSE, the default).

expand The default expansion factor (area inflation factor) for expansion of the simulation window in rmh (see rmhcontrol). Initialised to 2.

expand.polynom Logical. Whether expressions involving polynom in a model formula should be expanded, so that polynom(x,2) is replaced by x + I(x^2) and so on. Initialised to TRUE.

fastpois Logical. Whether to use a fast algorithm (introduced in **spatstat 1.42-3**) for simulating the Poisson point process in rpoispp when the argument lambda is a pixel image. Initialised to TRUE. Should be set to FALSE if needed to guarantee repeatability of results computed using earlier versions of **spatstat**.

fastthin Logical. Whether to use a fast C language algorithm (introduced in **spatstat 1.42-3**) for random thinning in rthin when the argument P is a single number. Initialised to TRUE. Should be set to FALSE if needed to guarantee repeatability of results computed using earlier versions of **spatstat**.

fastK.lgcp Logical. Whether to use fast or slow algorithm to compute the (theoretical) K-function of a log-Gaussian Cox process for use in lgcp.estK or Kmodel. The slow algorithm uses accurate numerical integration; the fast algorithm uses Simpson's Rule for numerical integration, and is about two orders of magnitude faster. Initialised to FALSE.

fixpolygons Logical flag indicating whether the functions owin and as.owin should repair errors in polygon data. For example, self-intersecting polygons and overlapping polygons will be repaired. The default is fixpolygons=TRUE.

fftw Logical value indicating whether the two-dimensional Fast Fourier Transform should be computed using the package **fftwtools**, instead of the fft function in the **stats** package. This affects the speed of density.ppp, density.psp, blur setcov and Smooth.ppp.

gpclib Defunct. This parameter was used to permit or forbid the use of the package **gpclib**, because of its restricted software licence. This package is no longer needed.

huge.npoints The maximum value of n for which runif(n) will not generate an error (possible errors include failure to allocate sufficient memory, and integer overflow of n). An attempt to generate more than this number of random points triggers a warning from runifpoint and other functions. Defaults to 1e6.

image.colfun Function determining the default colour map for plot.im. When called with one integer argument n, this function should return a character vector of length n specifying n different colours.

Kcom.remove.zeroes Logical value, determining whether the algorithm in Kcom and Kres removes or retains the contributions to the function from pairs of points that are identical. If these are retained then the function has a jump at $r = 0$. Initialised to TRUE.

maxedgewt Edge correction weights will be trimmed so as not to exceed this value. This applies to the weights computed by edge.Trans or edge.Ripley and used in Kest and its relatives.

maxmatrix The maximum permitted size (rows times columns) of matrices generated by **spatstat**'s internal code. Used by ppm and predict.ppm (for example) to decide when to split a large calculation into blocks. Defaults to 2^24=16777216.

monochrome Logical flag indicating whether graphics should be plotted in grey scale (monochrome=TRUE) or in colour (monochrome=FALSE, the default).

n.bandwidth Integer. Number of trial values of smoothing bandwidth to use for cross-validation in bw.relrisk and similar functions.

ndummy.min The minimum number of dummy points in a quadrature scheme created by `default.dummy`. Either an integer or a pair of integers giving the minimum number of dummy points in the `x` and `y` directions respectively.

ngrid.disc Number of points in the square grid used to compute a discrete approximation to the areas of discs in `areaLoss` and `areaGain` when exact calculation is not available. A single integer.

npixel Default number of pixels in a binary mask or pixel image. Either an integer, or a pair of integers, giving the number of pixels in the `x` and `y` directions respectively.

nvoxel Default number of voxels in a 3D image, typically for calculating the distance transform in `F3est`. Initialised to 4 megavoxels: `nvoxel = 2^22 = 4194304`.

par.binary List of arguments to be passed to the function `image` when displaying a binary image mask (in `plot.owin` or `plot.ppp`). Typically used to reset the colours of foreground and background.

par.contour List of arguments controlling contour plots of pixel images by `contour.im`.

par.fv List of arguments controlling the plotting of functions by `plot.fv` and its relatives.

par.persp List of arguments to be passed to the function `persp` when displaying a real-valued image, such as the fitted surfaces in `plot.ppm`.

par.points List of arguments controlling the plotting of point patterns by `plot.ppp`.

par.pp3 List of arguments controlling the plotting of three-dimensional point patterns by `plot.pp3`.

print.ppm.SE Default rule used by `print.ppm` to decide whether to calculate and print standard errors of the estimated coefficients of the model. One of the strings `"always"`, `"never"` or `"poisson"` (the latter indicating that standard errors will be calculated only for Poisson models). The default is `"poisson"` because the calculation for non-Poisson models can take a long time.

progress Character string determining the style of progress reports printed by `progressreport`. Either `"tty"`, `"tk"` or `"txtbar"`. For explanation of these options, see `progressreport`.

project.fast Logical. If `TRUE`, the algorithm of `project.ppm` will be accelerated using a shorcut. Initialised to `FALSE`.

psstA.ngrid Single integer, controlling the accuracy of the discrete approximation of areas computed in the function `psstA`. The area of a disc is approximated by counting points on an $n \times n$ grid. Initialised to 32.

psstA.nr Single integer, determining the number of distances r at which the function `psstA` will be evaluated (in the default case where argument `r` is absent). Initialised to 30.

psstG.remove.zeroes Logical value, determining whether the algorithm in `psstG` removes or retains the contributions to the function from pairs of points that are identical. If these are retained then the function has a jump at $r = 0$. Initialised to `TRUE`.

rmh.p, rmh.q, rmh.nrep New default values for the parameters `p`, `q` and `nrep` in the Metropolis-Hastings simulation algorithm. These override the defaults in `rmhcontrol.default`.

scalable Logical flag indicating whether the new code in `rmh.default` which makes the results scalable (invariant to change of units) should be used. In order to recover former behaviour (so that previous results can be reproduced) set this option equal to `FALSE`. See the "Warning" section in the help for `rmh()` for more detail.

terse Integer between 0 and 4. The level of terseness (brevity) in printed output from many functions in **spatstat**. Higher values mean shorter output. A rough guide is the following:

0	Full output
1	Avoid wasteful output
2	Remove space between paragraphs
3	Suppress extras such as standard errors
4	Compress text, suppress internal warnings

The value of `terse` is initialised to 0.

transparent Logical value indicating whether default colour maps are allowed to include semi-transparent colours, where possible. Default is `TRUE`. Currently this only affects `plot.ppp`.

units.paren The kind of parenthesis which encloses the text that explains a `unitname`. This text is seen in the text output of functions like `print.ppp` and in the graphics generated by `plot.fv`. The value should be one of the character strings `'('`, `'['`, `'{'` or `''`. The default is `'('`.

If no arguments are given, the current values of all parameters are returned, in a list.

If one parameter name is given, the current value of this parameter is returned (**not** in a list, just the value).

If several parameter names are given, the current values of these parameters are returned, in a list.

If `name=value` pairs are given, the named parameters are reset to the given values, and the **previous** values of these parameters are returned, in a list.

Value

Either a list of parameters and their values, or a single value. See Details.

Internal parameters

The following parameters may also be specified to `spatstat.options` but are intended for software development or testing purposes.

closepairs.newcode Logical. Whether to use new version of the code for `closepairs`. Initialised to `TRUE`.

crossing.psp.useCall Logical. Whether to use new version of the code for `crossing.psp`. Initialised to `TRUE`.

crosspairs.newcode Logical. Whether to use new version of the code for `crosspairs`. Initialised to `TRUE`.

densityC Logical. Indicates whether to use accelerated C code (`densityC=TRUE`) or interpreted R code (`densityC=FALSE`) to evaluate `density.ppp(X, at="points")`. Initialised to `TRUE`.

exactdt.checks.data Logical. Do not change this value, unless you are Adrian Baddeley `<Adrian.Baddeley@curtin.e`

fasteval One of the strings `'off'`, `'on'` or `'test'` determining whether to use accelerated C code to evaluate the conditional intensity of a Gibbs model. Initialised to `'on'`.

old.morpho.psp Logical. Whether to use old R code for morphological operations. Initialise to `FALSE`.

selfcrossing.psp.useCall Logical. Whether to use new version of the code for `selfcrossing.psp`. Initialised to `TRUE`.

use.Krect Logical. Whether to use new code for the K-function in a rectangular window. Initialised to `TRUE`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`options`

Examples

```
# save current values
oldopt <- spatstat.options()

spatstat.options("npixel")
spatstat.options(npixel=150)
spatstat.options(npixel=c(100,200))

spatstat.options(par.binary=list(col=grey(c(0.5,1))))

spatstat.options(par.persp=list(theta=-30,phi=40,d=4))
# see help(persp.default) for other options

# revert
spatstat.options(oldopt)
```

---

split.hyperframe *Divide Hyperframe Into Subsets and Reassemble*

Description

split divides the data x into subsets defined by f. The replacement form replaces values corresponding to such a division.

Usage

```
  ## S3 method for class 'hyperframe'
split(x, f, drop = FALSE, ...)

  ## S3 replacement method for class 'hyperframe'
split(x, f, drop = FALSE, ...) <- value
```

Arguments

x	Hyperframe (object of class "hyperframe").
f	a factor in the sense that as.factor(f) defines the grouping, or a list of such factors in which case their interaction is used for the grouping.
drop	logical value, indicating whether levels that do not occur should be dropped from the result.
value	a list of hyperframes which arose (or could have arisen) from the command split(x,f,drop=drop).
...	Ignored.

Details

These are methods for the generic functions split and split<- for hyperframes (objects of class "hyperframe").

A hyperframe is like a data frame, except that its entries can be objects of any kind. The behaviour of these methods is analogous to the corresponding methods for data frames.

Value

The value returned from `split.hyperframe` is a list of hyperframe containing the values for the groups. The components of the list are named by the levels of `f` (after converting to a factor, or if already a factor and `drop = TRUE`, dropping unused levels).

The replacement method `split<-.hyperframe` returns a new hyperframe `x` for which `split(x,f)` equals `value`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`hyperframe`, `[.hyperframe`

Examples

```
split(pyramidal, pyramidal$group)
```

`split.im` *Divide Image Into Sub-images*

Description

Divides a pixel image into several sub-images according to the value of a factor, or according to the tiles of a tessellation.

Usage

```
## S3 method for class 'im'
split(x, f, ..., drop = FALSE)
```

Arguments

`x`	Pixel image (object of class `"im"`).
`f`	Splitting criterion. Either a tessellation (object of class `"tess"`) or a pixel image with factor values.
`...`	Ignored.
`drop`	Logical value determining whether each subset should be returned as a pixel images (`drop=FALSE`) or as a one-dimensional vector of pixel values (`drop=TRUE`).

Details

This is a method for the generic function `split` for the class of pixel images. The image `x` will be divided into subsets determined by the data `f`. The result is a list of these subsets.

The splitting criterion may be either

- a tessellation (object of class `"tess"`). Each tile of the tessellation delineates a subset of the spatial domain.
- a pixel image (object of class `"im"`) with factor values. The levels of the factor determine subsets of the spatial domain.

If `drop=FALSE` (the default), the result is a list of pixel images, each one a subset of the pixel image `x`, obtained by restricting the pixel domain to one of the subsets. If `drop=TRUE`, then the pixel values are returned as numeric vectors.

Value

If `drop=FALSE`, a list of pixel images (objects of class `"im"`). It is also of class `"solist"` so that it can be plotted immediately.

If `drop=TRUE`, a list of numeric vectors.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`by.im`, `tess`, `im`

Examples

```
W <- square(1)
X <- as.im(function(x,y){sqrt(x^2+y^2)}, W)
Y <- dirichlet(runifpoint(12, W))
plot(split(X,Y))
```

`split.msr` *Divide a Measure into Parts*

Description

Decomposes a measure into components, each component being a measure.

Usage

```
  ## S3 method for class 'msr'
split(x, f, drop = FALSE, ...)
```

Arguments

x	Measure (object of class "msr") to be decomposed.
f	Factor or tessellation determining the decomposition. Argument passed to split.ppp. See Details.
drop	Logical value indicating whether empty components should be retained in the list (drop=FALSE, the default) or deleted (drop=TRUE).
...	Ignored.

Details

An object of class "msr" represents a signed (i.e. real-valued) or vector-valued measure in the **spatstat** package. See msr for explanation.

This function is a method for the generic split. It divides the measure x into components, each of which is a measure.

A measure x is represented in **spatstat** by a finite set of sample points with values attached to them. The function split.msr divides this pattern of sample points into several sub-patterns of points using split.ppp. For each sub-pattern, the values attached to these points are extracted from x, and these values and sample points determine a measure, which is a component or piece of the original x.

The argument f can be missing, if the sample points of x are multitype points. In this case, x represents a measure associated with marked spatial locations, and the command split(x) separates x into a list of component measures, one for each possible mark.

Otherwise the argument f is passed to split.ppp. It should be either a factor (of length equal to the number of sample points of x) or a tessellation (object of class "tess" representing a division of space into tiles) as documented under split.ppp.

Value

A list, each of whose entries is a measure (object of class "msr").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

msr, [.msr, with.msr

Examples

```
## split by tessellation
a <- residuals(ppm(cells ~ x))
aa <- split(a, dirichlet(runifpoint(4)))
aa
sapply(aa, integral)

## split by type of point
b <- residuals(ppm(amacrine ~ marks + x))
bb <- split(b)
bb
```

split.ppp *Divide Point Pattern into Sub-patterns*

Description

Divides a point pattern into several sub-patterns, according to their marks, or according to any user-specified grouping.

Usage

```
  ## S3 method for class 'ppp'
split(x, f = marks(x), drop=FALSE, un=NULL, reduce=FALSE, ...)
  ## S3 replacement method for class 'ppp'
split(x, f = marks(x), drop=FALSE, un=missing(f), ...) <- value
```

Arguments

`x`	A two-dimensional point pattern. An object of class "ppp".
`f`	Data determining the grouping. Either a factor, a logical vector, a pixel image with factor values, a tessellation, a window, or the name of one of the columns of marks.
`drop`	Logical. Determines whether empty groups will be deleted.
`un`	Logical. Determines whether the resulting subpatterns will be unmarked (i.e. whether marks will be removed from the points in each subpattern).
`reduce`	Logical. Determines whether to delete the column of marks used to split the pattern, when the marks are a data frame.
`...`	Other arguments are ignored.
`value`	List of point patterns.

Details

The function `split.ppp` divides up the points of the point pattern `x` into several sub-patterns according to the values of `f`. The result is a list of point patterns.

The argument `f` may be

- a factor, of length equal to the number of points in `x`. The levels of `f` determine the destination of each point in `x`. The `i`th point of `x` will be placed in the sub-pattern `split.ppp(x)$l` where `l = f[i]`.
- a pixel image (object of class "im") with factor values. The pixel value of `f` at each point of `x` will be used as the classifying variable.
- a tessellation (object of class "tess"). Each point of `x` will be classified according to the tile of the tessellation into which it falls.
- a window (object of class "owin"). Each point of `x` will be classified according to whether it falls inside or outside this window.
- a character string, matching the name of one of the columns of marks, if `marks(x)` is a data frame. This column should be a factor.

If `f` is missing, then it will be determined by the marks of the point pattern. The pattern `x` can be either

- a multitype point pattern (a marked point pattern whose marks vector is a factor). Then f is taken to be the marks vector. The effect is that the points of each type are separated into different point patterns.
- a marked point pattern with a data frame of marks, containing at least one column that is a factor. The first such column will be used to determine the splitting factor f.

Some of the sub-patterns created by the split may be empty. If drop=TRUE, then empty sub-patterns will be deleted from the list. If drop=FALSE then they are retained.

The argument un determines how to handle marks in the case where x is a marked point pattern. If un=TRUE then the marks of the points will be discarded when they are split into groups, while if un=FALSE then the marks will be retained.

If f and un are both missing, then the default is un=TRUE for multitype point patterns and un=FALSE for marked point patterns with a data frame of marks.

If the marks of x are a data frame, then split(x, reduce=TRUE) will discard only the column of marks that was used to split the pattern. This applies only when the argument f is missing.

The result of split.ppp has class "splitppp" and can be plotted using plot.splitppp.

The assignment function split<-.ppp updates the point pattern x so that it satisfies split(x, f, drop, un) = value. The argument value is expected to be a list of point patterns, one for each level of f. These point patterns are expected to be compatible with the type of data in the original pattern x.

Splitting can also be undone by the function superimpose, but this typically changes the ordering of the data.

Value

The value of split.ppp is a list of point patterns. The components of the list are named by the levels of f. The list also has the class "splitppp".

The assignment form split<-.ppp returns the updated point pattern x.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

cut.ppp, plot.splitppp, superimpose, im, tess, ppp.object

Examples

```
# (1) Splitting by marks

# Multitype point pattern: separate into types
 u <- split(amacrine)

# plot them
 plot(split(amacrine))

# the following are equivalent:
 amon <- split(amacrine)$on
 amon <- unmark(amacrine[amacrine$marks == "on"])
 amon <- subset(amacrine, marks == "on", -marks)
```

```
# the following are equivalent:
 amon <- split(amacrine, un=FALSE)$on
 amon <- amacrine[amacrine$marks == "on"]

# Scramble the locations of the 'on' cells
 X <- amacrine
 u <- split(X)
 u$on <- runifpoint(ex=amon)
 split(X) <- u

# Point pattern with continuous marks
 trees <- longleaf

 # cut the range of tree diameters into three intervals
 # using cut.ppp
 long3 <- cut(trees, breaks=3)
 # now split them
 long3split <- split(long3)

# (2) Splitting by a factor

# Unmarked point pattern
  swedishpines
# cut & split according to nearest neighbour distance
  f <- cut(nndist(swedishpines), 3)
  u <- split(swedishpines, f)

# (3) Splitting over a tessellation
   tes <- tess(xgrid=seq(0,96,length=5),ygrid=seq(0,100,length=5))
   v <- split(swedishpines, tes)


# (4) how to apply an operation to selected points:
#  split into components, transform desired component, then un-split
#  e.g. apply random jitter to 'on' points only
  X <- amacrine
  Y <- split(X)
  Y$on <- rjitter(Y$on, 0.1)
  split(X) <- Y
```

---

split.ppx *Divide Multidimensional Point Pattern into Sub-patterns*

Description

Divides a multidimensional point pattern into several sub-patterns, according to their marks, or according to any user-specified grouping.

Usage

```
  ## S3 method for class 'ppx'
split(x, f = marks(x), drop=FALSE, un=NULL, ...)
```

Arguments

x	A multi-dimensional point pattern. An object of class "ppx".
f	Data determining the grouping. Either a factor, or the name of one of the columns of marks.
drop	Logical. Determines whether empty groups will be deleted.
un	Logical. Determines whether the resulting subpatterns will be unmarked (i.e. whether marks will be removed from the points in each subpattern).
...	Other arguments are ignored.

Details

The generic command split allows a dataset to be separated into subsets according to the value of a grouping variable.

The function split.ppx is a method for the generic split for the class "ppx" of multidimensional point patterns. It divides up the points of the point pattern x into several sub-patterns according to the values of f. The result is a list of point patterns.

The argument f may be

- a factor, of length equal to the number of points in x. The levels of f determine the destination of each point in x. The ith point of x will be placed in the sub-pattern split.ppx(x)$l where l = f[i].
- a character string, matching the name of one of the columns of marks, if marks(x) is a data frame. This column should be a factor.

If f is missing, then it will be determined by the marks of the point pattern. The pattern x can be either

- a multitype point pattern (a marked point pattern whose marks vector is a factor). Then f is taken to be the marks vector. The effect is that the points of each type are separated into different point patterns.
- a marked point pattern with a data frame or hyperframe of marks, containing at least one column that is a factor. The first such column will be used to determine the splitting factor f.

Some of the sub-patterns created by the split may be empty. If drop=TRUE, then empty sub-patterns will be deleted from the list. If drop=FALSE then they are retained.

The argument un determines how to handle marks in the case where x is a marked point pattern. If un=TRUE then the marks of the points will be discarded when they are split into groups, while if un=FALSE then the marks will be retained.

If f and un are both missing, then the default is un=TRUE for multitype point patterns and un=FALSE for marked point patterns with a data frame of marks.

The result of split.ppx has class "splitppx" and "anylist". There are methods for print, summary and plot.

Value

A list of point patterns. The components of the list are named by the levels of f. The list also has the class "splitppx" and "anylist".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppx, plot.anylist

Examples

```
df <- data.frame(x=runif(4),y=runif(4),t=runif(4),
                 age=rep(c("old", "new"), 2),
                 size=runif(4))
X <- ppx(data=df, coord.type=c("s","s","t","m","m"))
X
split(X)
```

---

spokes *Spokes pattern of dummy points*

Description

Generates a pattern of dummy points in a window, given a data point pattern. The dummy points lie on the radii of circles emanating from each data point.

Usage

```
spokes(x, y, nrad = 3, nper = 3, fctr = 1.5, Mdefault = 1)
```

Arguments

x	Vector of x coordinates of data points, or a list with components x and y, or a point pattern (an object of class ppp).
y	Vector of y coordinates of data points. Ignored unless x is a vector.
nrad	Number of radii emanating from each data point.
nper	Number of dummy points per radius.
fctr	Scale factor. Length of largest spoke radius is fctr * M where M is the mean nearest neighbour distance for the data points.
Mdefault	Value of M to be used if x has length 1.

Details

This function is useful in creating dummy points for quadrature schemes (see quadscheme).

Given the data points, the function creates a collection of nrad * nper * length(x) dummy points.

Around each data point (x[i],y[i]) there are nrad * nper dummy points, lying on nrad radii emanating from (x[i],y[i]), with nper dummy points equally spaced along each radius.

The (equal) spacing of dummy points along each radius is controlled by the factor fctr. The distance from a data point to the furthest of its associated dummy points is fctr * M where M is the mean nearest neighbour distance for the data points.

If there is only one data point the nearest neighbour distance is infinite, so the value Mdefault will be used in place of M.

If x is a point pattern, then the value returned is also a point pattern, which is clipped to the window of x. Hence there may be fewer than nrad * nper * length(x) dummy points in the pattern returned.

Value

If argument x is a point pattern, a point pattern with window equal to that of x. Otherwise a list with two components x and y. In either case the components x and y of the value are numeric vectors giving the coordinates of the dummy points.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quad.object, quadscheme, inside.owin, gridcentres, stratrand

Examples

```
dat <- runifrect(10)
dum <- spokes(dat$x, dat$y, 5, 3, 0.7)
plot(dum)
Q <- quadscheme(dat, dum, method="dirichlet")
plot(Q, tiles=TRUE)
```

---

square *Square Window*

Description

Creates a square window

Usage

```
square(r=1, unitname=NULL)
unit.square()
```

Arguments

r	Numeric. The side length of the square, or a vector giving the minimum and maximum coordinate values.
unitname	Optional. Name of unit of length. Either a single character string, or a vector of two character strings giving the singular and plural forms, respectively.

Details

If r is a number, square(r) is a shortcut for creating a window object representing the square $[0, r] \times [0, r]$. It is equivalent to the command owin(c(0,r),c(0,r)).

If r is a vector of length 2, then square(r) creates the square with x and y coordinates ranging from r[1] to r[2].

unit.square creates the unit square $[0, 1] \times [0, 1]$. It is equivalent to square(1) or square() or owin(c(0,1),c(0,1)).

These commands are included for convenience, and to improve the readability of some code.

Value

An object of class "owin" (see owin.object) specifying a window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

owin.object, owin

Examples

```
W <- square(10)
W <- square(c(-1,1))
```

---

ssf *Spatially Sampled Function*

Description

Create an object that represents a spatial function which has been evaluated or sampled at an irregular set of points.

Usage

```
ssf(loc, val)
```

Arguments

loc	The spatial locations at which the function has been evaluated. A point pattern (object of class "ppp").
val	The function values at these locations. A numeric vector with one entry for each point of loc, or a data frame with one row for each point of loc.

Details

An object of class "ssf" represents a real-valued or vector-valued function that has been evaluated or sampled at an irregular set of points. An example would be a spatial covariate that has only been measured at certain locations.

An object of this class also inherits the class "ppp", and is essentially the same as a marked point pattern, except for the class membership which enables it to be handled in a different way.

There are methods for plot, print etc; see plot.ssf and methods.ssf.

Use unmark to extract only the point locations, and marks.ssf to extract only the function values.

Value

Object of class "ssf".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

plot.ssf, methods.ssf, Smooth.ssf, with.ssf, [.ssf.

Examples

```
ssf(cells, nndist(cells, k=1:3))
```

---

stieltjes *Compute Integral of Function Against Cumulative Distribution*

Description

Computes the Stieltjes integral of a function f with respect to a function M.

Usage

```
stieltjes(f, M, ...)
```

Arguments

f	The integrand. A function in the R language.
M	The cumulative function against which f will be integrated. An object of class "fv" or "stepfun".
...	Additional arguments passed to f.

Details

This command computes the Stieltjes integral

$$I = \int f(x) dM(x)$$

of a real-valued function $f(x)$ with respect to a nondecreasing function $M(x)$.

One common use of the Stieltjes integral is to find the mean value of a random variable from its cumulative distribution function $F(x)$. The mean value is the Stieltjes integral of $f(x) = x$ with respect to $F(x)$.

The argument f should be a function in the R language. It should accept a numeric vector argument x and should return a numeric vector of the same length.

The argument M should be either a step function (object of class "stepfun") or a function value table (object of class "fv", see fv.object). Objects of class "stepfun" are returned by ecdf, ewcdf, spatialcdf and other utilities. Objects of class "fv" are returned by the commands Kest, Gest, etc.

Value

A list containing the value of the Stieltjes integral computed using each of the versions of the function M.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

fv.object, Gest

Examples

```
# estimate cdf of nearest neighbour distance in redwood data
G <- Gest(redwood)
# compute estimate of mean nearest neighbour distance
stieltjes(function(x){x}, G)
# estimated probability of a distance in the interval [0.1,0.2]
stieltjes(function(x,a,b){ (x >= a) & (x <= b)}, G, a=0.1, b=0.2)

# stepfun example
H <- spatialcdf(bei.extra$elev, normalise=TRUE)
stieltjes(function(x){x}, H)
```

---

stienen *Stienen Diagram*

Description

Draw the Stienen diagram of a point pattern, or compute the region covered by the Stienen diagram.

Usage

```
stienen(X, ..., bg = "grey", border = list(bg = NULL))
stienenSet(X, edge=TRUE)
```

Arguments

X	Point pattern (object of class "ppp").
...	Arguments passed to plot.ppp to control the plot.
bg	Fill colour for circles.
border	Either a list of arguments passed to plot.ppp to control the display of circles at the border of the diagram, or the value FALSE indicating that the border circles should not be plotted.
edge	Logical value indicating whether to include the circles at the border of the diagram.

Details

The Stienen diagram of a point pattern (Stienen, 1982) is formed by drawing a circle around each point of the pattern, with diameter equal to the nearest-neighbour distance for that point. These circles do not overlap. If two points are nearest neighbours of each other, then the corresponding circles touch.

`stienenSet(X)` computes the union of these circles and returns it as a window (object of class `"owin"`).

`stienen(X)` generates a plot of the Stienen diagram of the point pattern `X`. By default, circles are shaded in grey if they lie inside the window of `X`, and are not shaded otherwise.

Value

The plotting function `stienen` returns `NULL`.

The return value of `stienenSet` is a window (object of class `"owin"`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Stienen, H. (1982) *Die Vergroeberung von Karbiden in reinen Eisen-Kohlenstoff Staehlen*. Dissertation, RWTH Aachen.

See Also

`nndist`, `plot.ppp`

Examples

```
Y <- stienenSet(cells)
stienen(redwood)
stienen(redwood, border=list(bg=NULL, lwd=2, cols="red"))
```

`stratrand` *Stratified random point pattern*

Description

Generates a “stratified random” pattern of points in a window, by dividing the window into rectangular tiles and placing `k` random points in each tile.

Usage

```
stratrand(window, nx, ny, k = 1)
```

Arguments

window	A window. An object of class owin, or data in any format acceptable to as.owin().
nx	Number of tiles in each row.
ny	Number of tiles in each column.
k	Number of random points to generate in each tile.

Details

The bounding rectangle of window is divided into a regular $nx \times ny$ grid of rectangular tiles. In each tile, k random points are generated independently with a uniform distribution in that tile.

Note that some of these grid points may lie outside the window, if window is not of type "rectangle". The function inside.owin can be used to select those grid points which do lie inside the window. See the examples.

This function is useful in creating dummy points for quadrature schemes (see quadscheme) as well as in simulating random point patterns.

Value

A list with two components x and y, which are numeric vectors giving the coordinates of the random points.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quad.object, quadscheme, inside.owin, gridcentres

Examples

```
  w <- unit.square()
  xy <- stratrand(w, 10, 10)
  ## Not run:
  plot(w)
  points(xy)

## End(Not run)

  # polygonal boundary
  bdry <- list(x=c(0.1,0.3,0.7,0.4,0.2),
               y=c(0.1,0.1,0.5,0.7,0.3))
  w <- owin(c(0,1), c(0,1), poly=bdry)
  xy <- stratrand(w, 10, 10, 3)
  ## Not run:
  plot(w)
  points(xy)

## End(Not run)
  # determine which grid points are inside polygon
  ok <- inside.owin(xy$x, xy$y, w)
  ## Not run:
```

```
plot(w)
points(xy$x[ok], xy$y[ok])

## End(Not run)
```

Strauss *The Strauss Point Process Model*

Description

Creates an instance of the Strauss point process model which can then be fitted to point pattern data.

Usage

```
Strauss(r)
```

Arguments

r The interaction radius of the Strauss process

Details

The (stationary) Strauss process with interaction radius r and parameters β and γ is the pairwise interaction point process in which each point contributes a factor β to the probability density of the point pattern, and each pair of points closer than r units apart contributes a factor γ to the density.

Thus the probability density is

$$f(x_1, \ldots, x_n) = \alpha \beta^{n(x)} \gamma^{s(x)}$$

where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, $s(x)$ is the number of distinct unordered pairs of points that are closer than r units apart, and α is the normalising constant.

The interaction parameter γ must be less than or equal to 1 so that this model describes an "ordered" or "inhibitive" pattern.

The nonstationary Strauss process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location, rather than a constant beta.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Strauss process pairwise interaction is yielded by the function `Strauss()`. See the examples below.

Note the only argument is the interaction radius `r`. When `r` is fixed, the model becomes an exponential family. The canonical parameters $\log(\beta)$ and $\log(\gamma)$ are estimated by `ppm()`, not fixed in `Strauss()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the Strauss process with interaction radius r.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Kelly, F.P. and Ripley, B.D. (1976) On Strauss's model for clustering. *Biometrika* **63**, 357–360.

Strauss, D.J. (1975) A model for clustering. *Biometrika* **62**, 467–475.

See Also

ppm, pairwise.family, ppm.object

Examples

```
   Strauss(r=0.1)
   # prints a sensible description of itself
   data(cells)

   ## Not run:
   ppm(cells, ~1, Strauss(r=0.07))
   # fit the stationary Strauss process to `cells'

## End(Not run)


   ppm(cells, ~polynom(x,y,3), Strauss(r=0.07))
   # fit a nonstationary Strauss process with log-cubic polynomial trend
```

StraussHard — *The Strauss / Hard Core Point Process Model*

Description

Creates an instance of the "Strauss/ hard core" point process model which can then be fitted to point pattern data.

Usage

```
StraussHard(r, hc=NA)
```

Arguments

r	The interaction radius of the Strauss interaction
hc	The hard core distance. Optional.

Details

A Strauss/hard core process with interaction radius r, hard core distance $h < r$, and parameters β and γ, is a pairwise interaction point process in which

- distinct points are not allowed to come closer than a distance h apart
- each pair of points closer than r units apart contributes a factor γ to the probability density.

This is a hybrid of the Strauss process and the hard core process.

The probability density is zero if any pair of points is closer than h units apart, and otherwise equals

$$f(x_1, \ldots, x_n) = \alpha\beta^{n(x)}\gamma^{s(x)}$$

where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, $s(x)$ is the number of distinct unordered pairs of points that are closer than r units apart, and α is the normalising constant.

The interaction parameter γ may take any positive value (unlike the case for the Strauss process). If $\gamma < 1$, the model describes an "ordered" or "inhibitive" pattern. If $\gamma > 1$, the model is "ordered" or "inhibitive" up to the distance h, but has an "attraction" between points lying at distances in the range between h and r.

If $\gamma = 1$, the process reduces to a classical hard core process with hard core distance h. If $\gamma = 0$, the process reduces to a classical hard core process with hard core distance r.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Strauss/hard core process pairwise interaction is yielded by the function `StraussHard()`. See the examples below.

The canonical parameter $\log(\gamma)$ is estimated by `ppm()`, not fixed in `StraussHard()`.

If the hard core distance argument `hc` is missing or `NA`, it will be estimated from the data when `ppm` is called. The estimated value of `hc` is the minimum nearest neighbour distance multiplied by $n/(n+1)$, where n is the number of data points.

Value

An object of class `"interact"` describing the interpoint interaction structure of the "Strauss/hard core" process with Strauss interaction radius r and hard core distance `hc`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

Strauss, D.J. (1975) A model for clustering. *Biometrika* **62**, 467–475.

See Also

`ppm`, `pairwise.family`, `ppm.object`

Examples

```
StraussHard(r=1,hc=0.02)
# prints a sensible description of itself

data(cells)

## Not run:
ppm(cells, ~1, StraussHard(r=0.1, hc=0.05))
# fit the stationary Strauss/hard core  process to `cells'

## End(Not run)

ppm(cells, ~ polynom(x,y,3), StraussHard(r=0.1, hc=0.05))
# fit a nonstationary Strauss/hard core process
# with log-cubic polynomial trend
```

---

studpermu.test *Studentised Permutation Test*

Description

Perform a studentised permutation test for a difference between groups of point patterns.

Usage

```
studpermu.test(X, formula, summaryfunction = Kest,
    ..., rinterval = NULL, nperm = 999,
     use.Tbar = FALSE, minpoints = 20, rsteps = 128,
     r = NULL, arguments.in.data = FALSE)
```

Arguments

X	Data. Either a hyperframe or a list of lists of point patterns.
formula	Formula describing the grouping, when X is a hyperframe. The left side of the formula identifies which column of X contains the point patterns. The right side identifies the grouping factor. If the formula is missing, the grouping variable is taken to be the first column of X that contains a factor, and the point patterns are taken from the first column that contains point patterns.
summaryfunction	Summary function applicable to point patterns.
...	Additional arguments passed to summaryfunction.
rinterval	Interval of distance values r over which the summary function should be evaluated and over which the test statistic will be integrated. If NULL, the default range of the summary statistic is used (taking the intersection of these ranges over all patterns).
nperm	Number of random permutations for the test.
use.Tbar	Logical value indicating choice of test statistic. If TRUE, use the alternative test statistic, which is appropriate for summary functions with roughly constant variance, such as $K(r)/r$ or $L(r)$.

`minpoints` Minimum permissible number of points in a point pattern for inclusion in the test calculation.

`rsteps` Number of discretisation steps in the `rinterval`.

`r` Optional vector of distance values as the argument for `summaryfunction`. Should not usually be given. There is a sensible default.

`arguments.in.data` Logical. If `TRUE`, individual extra arguments to `summaryfunction` will be taken from `X` (which must be a hyperframe). This assumes that the first argument of `summaryfunction` is the point pattern dataset.

Details

This function performs the studentized permutation test of Hahn (2012) for a difference between groups of point patterns.

The first argument `X` should be either

a list of lists of point patterns. Each element of `X` will be interpreted as a group of point patterns, assumed to be replicates of the same point process.

a hyperframe: One column of the hyperframe should contain point patterns, and another column should contain a factor indicating the grouping. The argument `formula` should be a formula in the R language specifying the grouping: it should be of the form `P ~ G` where `P` is the name of the column of point patterns, and `G` is the name of the factor.

A group needs to contain at least two point patterns with at least `minpoints` points in each pattern.

The function returns an object of class `"htest"` and `"studpermutest"` that can be printed and plotted. The printout shows the test result and p-value. The plot shows the summary functions for the groups (and the group means if requested).

Value

Object of class `"studpermutest"`.

Author(s)

Ute Hahn.

Modified for `spatstat` by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Hahn, U. (2012) A studentized permutation test for the comparison of spatial point patterns. *Journal of the American Statistical Association* **107** (498), 754–764.

Examples

```
np <- if(interactive()) 99 else 19
testpyramidal <- studpermu.test(pyramidal, Neurons ~ group, nperm=np)
testpyramidal
```

subfits *Extract List of Individual Point Process Models*

Description

Takes a Gibbs point process model that has been fitted to several point patterns simultaneously, and produces a list of fitted point process models for the individual point patterns.

Usage

```
subfits(object, what="models", verbose=FALSE)
subfits.old(object, what="models", verbose=FALSE)
subfits.new(object, what="models", verbose=FALSE)
```

Arguments

object	An object of class "mppm" representing a point process model fitted to several point patterns.
what	What should be returned. Either "models" to return the fitted models, or "interactions" to return the fitted interactions only.
verbose	Logical flag indicating whether to print progress reports.

Details

object is assumed to have been generated by mppm. It represents a point process model that has been fitted to a list of several point patterns, with covariate data.

For each of the *individual* point pattern datasets, this function derives the corresponding fitted model for that dataset only (i.e. a point process model for the ith point pattern, that is consistent with object).

If what="models", the result is a list of point process models (a list of objects of class "ppm"), one model for each point pattern dataset in the original fit. If what="interactions", the result is a list of fitted interpoint interactions (a list of objects of class "fii").

Two different algorithms are provided, as subfits.old and subfits.new. Currently subfits is the same as the old algorithm subfits.old because the newer algorithm is too memory-hungry.

Value

A list of point process models (a list of objects of class "ppm") or a list of fitted interpoint interactions (a list of objects of class "fii").

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented in **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

See Also

mppm, ppm

Examples

```
H <- hyperframe(Wat=waterstriders)
fit <- mppm(Wat~x, data=H)
subfits(fit)

H$Wat[[3]] <- rthin(H$Wat[[3]], 0.1)
fit2 <- mppm(Wat~x, data=H, random=~1|id)
subfits(fit2)
```

---

subset.hyperframe *Subset of Hyperframe Satisfying A Condition*

Description

Given a hyperframe, return the subset specified by imposing a condition on each row, and optionally by choosing only some of the columns.

Usage

```
## S3 method for class 'hyperframe'
subset(x, subset, select, ...)
```

Arguments

x	A hyperframe pattern (object of class "hyperframe".
subset	Logical expression indicating which points are to be kept. The expression may involve the names of columns of x and will be evaluated by with.hyperframe.
select	Expression indicating which columns of marks should be kept.
...	Arguments passed to [.hyperframe such as drop and strip.

Details

This is a method for the generic function subset. It extracts the subset of rows of x that satisfy the logical expression subset, and retains only the columns of x that are specified by the expression select. The result is always a hyperframe.

The argument subset determines the subset of rows that will be extracted. It should be a logical expression. It may involve the names of columns of x. The default is to keep all points.

The argument select determines which columns of x will be retained. It should be an expression involving the names of columns (which will be interpreted as integers representing the positions of these columns). For example if there are columns named A to Z, then select=D:F is a valid expression and means that columns D, E and F will be retained. Similarly select=-(A:C) is valid and means that columns A to C will be deleted. The default is to retain all columns.

Setting subset=FALSE will remove all the rows. Setting select=FALSE will remove all the columns.

The result is always a hyperframe.

Value

A hyperframe.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

subset, [.hyperframe

Examples

```
a <- subset(flu, virustype=="wt")

aa <- subset(flu, minnndist(pattern) > 10)

aaa <- subset(flu, virustype=="wt", select = -pattern)
```

---

subset.ppp *Subset of Point Pattern Satisfying A Condition*

Description

Given a point pattern, return the subset of points which satisfy a specified condition.

Usage

```
## S3 method for class 'ppp'
subset(x, subset, select, drop=FALSE, ...)

## S3 method for class 'pp3'
subset(x, subset, select, drop=FALSE, ...)

## S3 method for class 'lpp'
subset(x, subset, select, drop=FALSE, ...)

## S3 method for class 'ppx'
subset(x, subset, select, drop=FALSE, ...)
```

Arguments

x	A point pattern (object of class "ppp", "lpp", "pp3" or "ppx").
subset	Logical expression indicating which points are to be kept. The expression may involve the names of spatial coordinates (x, y, etc), the marks, and (if there is more than one column of marks) the names of individual columns of marks. Missing values are taken as false. See Details.

select	Expression indicating which columns of marks should be kept. The *names* of columns of marks can be used in this expression, and will be treated as if they were column indices. See Details.
drop	Logical value indicating whether to remove unused levels of the marks, if the marks are a factor.
...	Ignored.

Details

This is a method for the generic function subset. It extracts the subset of points of x that satisfy the logical expression subset, and retains only the columns of marks that are specified by the expression select. The result is always a point pattern, with the same window as x.

The argument subset determines the subset of points that will be extracted. It should be a logical expression. It may involve the variable names x and y representing the Cartesian coordinates; the names of other spatial coordinates or local coordinates; the name marks representing the marks; and (if there is more than one column of marks) the names of individual columns of marks. The default is to keep all points.

The argument select determines which columns of marks will be retained (if there are several columns of marks). It should be an expression involving the names of columns of marks (which will be interpreted as integers representing the positions of these columns). For example if there are columns of marks named A to Z, then select=D:F is a valid expression and means that columns D, E and F will be retained. Similarly select=-(A:C) is valid and means that columns A to C will be deleted. The default is to retain all columns.

Setting subset=FALSE will produce an empty point pattern (i.e. containing zero points) in the same window as x. Setting select=FALSE or select= -marks will remove all the marks from x.

The argument drop determines whether to remove unused levels of a factor, if the resulting point pattern is multitype (i.e. the marks are a factor) or if the marks are a data frame in which some of the columns are factors.

The result is always a point pattern, of the same class as x. Spatial coordinates (and local coordinates) are always retained. To extract only some columns of marks or coordinates as a data frame, use subset(as.data.frame(x), ...)

Value

A point pattern of the same class as x, in the same spatial window as x. The result is a subset of x, possibly with some columns of marks removed.

Other kinds of subset arguments

Alternatively the argument subset can be any kind of subset index acceptable to [.ppp, [.pp3, [.ppx. This argument selects which points of x will be retained.

Warning: if the argument subset is a window, this is interpreted as specifying the subset of points that fall inside that window, but the resulting point pattern has the same window as the original pattern x.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

subset,
[.ppp, [.pp3, [.lpp, [.ppx

Examples

```
plot(subset(cells, x > 0.5))

subset(amacrine, marks == "on")

subset(amacrine, marks == "on", drop=TRUE)

subset(redwood, nndist(redwood) > 0.04)

subset(finpines, select=height)

subset(finpines, diameter > 2, height)

subset(nbfires, year==1999 & ign.src == "campfire",
                select=cause:fnl.size)

v <- subset(chicago, x + y > 1100 & marks == "assault")

vv <- subset(chicago, x + y > 1100 & marks == "assault", drop=TRUE)

a <- subset(rpoispp3(40), z > 0.5)
```

---

subspaceDistance *Distance Between Linear Spaces*

Description

Evaluate the distance between two linear subspaces using the measure proposed by Li, Zha and Chiaromonte (2005).

Usage

```
subspaceDistance(B0, B1)
```

Arguments

B0	Matrix whose columns are a basis for the first subspace.
B1	Matrix whose columns are a basis for the second subspace.

Details

This algorithm calculates the maximum absolute value of the eigenvalues of $P1-P0$ where $P0, P1$ are the projection matrices onto the subspaces generated by B0,B1. This measure of distance was proposed by Li, Zha and Chiaromonte (2005). See also Xia (2007).

Value

A single numeric value.

Author(s)

Matlab original by Yongtao Guan, translated to R by Suman Rakshit.

References

Guan, Y. and Wang, H. (2010) Sufficient dimension reduction for spatial point processes directed by Gaussian random fields. *Journal of the Royal Statistical Society, Series B*, **72**, 367–387.

Li, B., Zha, H. and Chiaromonte, F. (2005) Contour regression: a general approach to dimension reduction. *Annals of Statistics* **33**, 1580–1616.

Xia, Y. (2007) A constructive approach to the estimation of dimension reduction directions. *Annals of Statistics* **35**, 2654–2690.

---

suffstat *Sufficient Statistic of Point Process Model*

Description

The canonical sufficient statistic of a point process model is evaluated for a given point pattern.

Usage

```
suffstat(model, X=data.ppm(model))
```

Arguments

`model`	A fitted point process model (object of class "ppm").
`X`	A point pattern (object of class "ppp").

Details

The canonical sufficient statistic of `model` is evaluated for the point pattern `X`. This computation is useful for various Monte Carlo methods.

Here `model` should be a point process model (object of class "ppm", see `ppm.object`), typically obtained from the model-fitting function `ppm`. The argument `X` should be a point pattern (object of class "ppp").

Every point process model fitted by `ppm` has a probability density of the form

$$f(x) = Z(\theta) \exp(\theta^T S(x))$$

where x denotes a typical realisation (i.e. a point pattern), θ is the vector of model coefficients, $Z(\theta)$ is a normalising constant, and $S(x)$ is a function of the realisation x, called the "canonical sufficient statistic" of the model.

For example, the stationary Poisson process has canonical sufficient statistic $S(x) = n(x)$, the number of points in x. The stationary Strauss process with interaction range r (and fitted with no edge correction) has canonical sufficient statistic $S(x) = (n(x), s(x))$ where $s(x)$ is the number of pairs of points in x which are closer than a distance r to each other.

`suffstat(model, X)` returns the value of $S(x)$, where S is the canonical sufficient statistic associated with `model`, evaluated when x is the given point pattern `X`. The result is a numeric vector, with entries which correspond to the entries of the coefficient vector `coef(model)`.

The sufficient statistic S does not depend on the fitted coefficients of the model. However it does depend on the irregular parameters which are fixed in the original call to ppm, for example, the interaction range r of the Strauss process.

The sufficient statistic also depends on the edge correction that was used to fit the model. For example in a Strauss process,

- If the model is fitted with `correction="none"`, the sufficient statistic is $S(x) = (n(x), s(x))$ where $n(x)$ is the number of points and $s(x)$ is the number of pairs of points which are closer than r units apart.
- If the model is fitted with `correction="periodic"`, the sufficient statistic is the same as above, except that distances are measured in the periodic sense.
- If the model is fitted with `correction="translate"`, then $n(x)$ is unchanged but $s(x)$ is replaced by a weighted sum (the sum of the translation correction weights for all pairs of points which are closer than r units apart).
- If the model is fitted with `correction="border"` (the default), then points lying less than r units from the boundary of the observation window are treated as fixed. Thus $n(x)$ is replaced by the number $n_r(x)$ of points lying at least r units from the boundary of the observation window, and $s(x)$ is replaced by the number $s_r(x)$ of pairs of points, which are closer than r units apart, and at least one of which lies more than r units from the boundary of the observation window.

Non-finite values of the sufficient statistic (NA or -Inf) may be returned if the point pattern X is not a possible realisation of the model (i.e. if X has zero probability of occurring under model for all values of the canonical coefficients θ).

Value

A numeric vector of sufficient statistics. The entries correspond to the model coefficients `coef(model)`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

ppm

Examples

```
fitS <- ppm(swedishpines~1, Strauss(7))
X <- rpoispp(intensity(swedishpines), win=Window(swedishpines))
suffstat(fitS, X)
```

`summary.anylist` *Summary of a List of Things*

Description

Prints a useful summary of each item in a list of things.

Usage

```
## S3 method for class 'anylist'
summary(object, ...)
```

Arguments

`object`	An object of class "anylist".
`...`	Ignored.

Details

This is a method for the generic function `summary`.

An object of the class "anylist" is effectively a list of things which are intended to be treated in a similar way. See `anylist`.

This function extracts a useful summary of each of the items in the list.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`anylist`, `summary`, `plot.anylist`

Examples

```
x <- anylist(A=runif(10), B=runif(10), C=runif(10))
summary(x)
```

summary.im *Summarizing a Pixel Image*

Description

summary method for class "im".

Usage

```
## S3 method for class 'im'
summary(object, ...)
## S3 method for class 'summary.im'
print(x, ...)
```

Arguments

object	A pixel image.
...	Ignored.
x	Object of class "summary.im" as returned by summary.im.

Details

This is a method for the generic summary for the class "im". An object of class "im" describes a pixel image. See im.object) for details of this class.

summary.im extracts information about the pixel image, and print.summary.im prints this information in a comprehensible format.

In normal usage, print.summary.im is invoked implicitly when the user calls summary.im without assigning its value to anything. See the examples.

The information extracted by summary.im includes

range The range of the image values.

mean The mean of the image values.

integral The "integral" of the image values, calculated as the sum of the image values multiplied by the area of one pixel.

dim The dimensions of the pixel array: dim[1] is the number of rows in the array, corresponding to the **y** coordinate.

Value

summary.im returns an object of class "summary.im", while print.summary.im returns NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

mean.im, integral.im, anyNA.im

Examples

```
# make an image
X <- as.im(function(x,y) {x^2}, unit.square())
# summarize it
summary(X)
# save the summary
s <- summary(X)
# print it
print(X)
s
# extract stuff
X$dim
X$range
X$integral
```

---

summary.kppm *Summarizing a Fitted Cox or Cluster Point Process Model*

Description

summary method for class "kppm".

Usage

```
  ## S3 method for class 'kppm'
summary(object, ..., quick=FALSE)

  ## S3 method for class 'summary.kppm'
print(x, ...)
```

Arguments

object	A fitted Cox or cluster point process model (object of class "kppm").
quick	Logical value controlling the scope of the summary.
...	Arguments passed to summary.ppm or print.summary.ppm controlling the treatment of the trend component of the model.
x	Object of class "summary.kppm" as returned by summary.kppm.

Details

This is a method for the generic summary for the class "kppm". An object of class "kppm" describes a fitted Cox or cluster point process model. See kppm.

summary.kppm extracts information about the type of model that has been fitted, the data to which the model was fitted, and the values of the fitted coefficients.

print.summary.kppm prints this information in a comprehensible format.

In normal usage, print.summary.kppm is invoked implicitly when the user calls summary.kppm without assigning its value to anything. See the examples.

You can also type coef(summary(object)) to extract a table of the fitted coefficients of the point process model object together with standard errors and confidence limits.

Value

summary.kppm returns an object of class "summary.kppm", while print.summary.kppm returns NULL.

The result of summary.kppm includes at least the following components:

Xname	character string name of the original point pattern data
stationary	logical value indicating whether the model is stationary
clusters	the clusters argument to kppm
modelname	character string describing the model
isPCP	TRUE if the model is a Poisson cluster process, FALSE if it is a log-Gaussian Cox process
lambda	Estimated intensity: numeric value, or pixel image
mu	Mean cluster size: numeric value, pixel image, or NULL
clustpar	list of fitted parameters for the cluster model
clustargs	list of fixed parameters for the cluster model, if any
callstring	character string representing the original call to kppm

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

Examples

```
fit <- kppm(redwood ~ 1, "Thomas")
summary(fit)
coef(summary(fit))
```

---

summary.listof *Summary of a List of Things*

Description

Prints a useful summary of each item in a list of things.

Usage

```
 ## S3 method for class 'listof'
summary(object, ...)
```

Arguments

object	An object of class "listof".
...	Ignored.

Details

This is a method for the generic function summary.

An object of the class "listof" is effectively a list of things which are all of the same class.

This function extracts a useful summary of each of the items in the list.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

summary, plot.listof

Examples

```
x <- list(A=runif(10), B=runif(10), C=runif(10))
class(x) <- c("listof", class(x))
summary(x)
```

---

summary.owin *Summary of a Spatial Window*

Description

Prints a useful description of a window object.

Usage

```
## S3 method for class 'owin'
summary(object, ...)
```

Arguments

object	Window (object of class "owin").
...	Ignored.

Details

A useful description of the window object is printed.

This is a method for the generic function summary.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

summary, summary.ppp, print.owin

Examples

```
summary(owin())  # the unit square

data(demopat)
W <- Window(demopat)  # weird polygonal window
summary(W)           # describes it

summary(as.mask(W))  # demonstrates current pixel resolution
```

---

summary.ppm *Summarizing a Fitted Point Process Model*

Description

summary method for class "ppm".

Usage

```
  ## S3 method for class 'ppm'
summary(object, ..., quick=FALSE, fine=FALSE)
  ## S3 method for class 'summary.ppm'
print(x, ...)
```

Arguments

object	A fitted point process model.
...	Ignored.
quick	Logical flag controlling the scope of the summary.
fine	Logical value passed to vcov.ppm determining whether to compute the quick, coarse estimate of variance (fine=FALSE, the default) or the slower, finer estimate (fine=TRUE).
x	Object of class "summary.ppm" as returned by summary.ppm.

Details

This is a method for the generic summary for the class "ppm". An object of class "ppm" describes a fitted point process model. See ppm.object) for details of this class.

summary.ppm extracts information about the type of model that has been fitted, the data to which the model was fitted, and the values of the fitted coefficients. (If quick=TRUE then only the information about the type of model is extracted.)

print.summary.ppm prints this information in a comprehensible format.

In normal usage, print.summary.ppm is invoked implicitly when the user calls summary.ppm without assigning its value to anything. See the examples.

You can also type coef(summary(object)) to extract a table of the fitted coefficients of the point process model object together with standard errors and confidence limits.

Value

summary.ppm returns an object of class "summary.ppm", while print.summary.ppm returns NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
  # invent some data
  X <- rpoispp(42)
  # fit a model to it
  fit <- ppm(X ~ x, Strauss(r=0.1))
  # summarize the fitted model
  summary(fit)
  # `quick' option
  summary(fit, quick=TRUE)
  # coefficients with standard errors and CI
  coef(summary(fit))
  coef(summary(fit, fine=TRUE))

  # save the full summary
  s <- summary(fit)
  # print it
  print(s)
  s
  # extract stuff
  names(s)
  coef(s)
  s$args$correction
  s$name
  s$trend$value

  ## Not run:
  # multitype pattern
  data(demopat)
  fit <- ppm(demopat, ~marks, Poisson())
  summary(fit)

## End(Not run)

  # model with external covariates
  fitX <- ppm(X, ~Z, covariates=list(Z=function(x,y){x+y}))
  summary(fitX)
```

---

summary.ppp *Summary of a Point Pattern Dataset*

Description

Prints a useful summary of a point pattern dataset.

Usage

```
 ## S3 method for class 'ppp'
summary(object, ..., checkdup=TRUE)
```

Arguments

`object`	Point pattern (object of class "ppp").
`...`	Ignored.
`checkdup`	Logical value indicating whether to check for the presence of duplicate points.

Details

A useful summary of the point pattern `object` is printed.

This is a method for the generic function `summary`.

If `checkdup=TRUE`, the pattern will be checked for the presence of duplicate points, using `duplicated.ppp`. This can be time-consuming if the pattern contains many points, so the checking can be disabled by setting `checkdup=FALSE`.

If the point pattern was generated by simulation using `rmh`, the parameters of the algorithm are printed.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

`summary`, `summary.owin`, `print.ppp`

Examples

```
summary(cells)  # plain vanilla point pattern

# multitype point pattern
woods <- lansing

summary(woods) # tabulates frequencies of each mark

# numeric marks
trees <- longleaf

summary(trees) # prints summary.default(marks(trees))

# weird polygonal window
summary(demopat)  # describes it
```

`summary.psp` *Summary of a Line Segment Pattern Dataset*

Description

Prints a useful summary of a line segment pattern dataset.

Usage

```
  ## S3 method for class 'psp'
summary(object, ...)
```

Arguments

object	Line segment pattern (object of class "psp").
...	Ignored.

Details

A useful summary of the line segment pattern object is printed.

This is a method for the generic function summary.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

summary, summary.owin, print.psp

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
summary(a)  # describes it
```

---

summary.quad *Summarizing a Quadrature Scheme*

Description

summary method for class "quad".

Usage

```
  ## S3 method for class 'quad'
summary(object, ..., checkdup=FALSE)
  ## S3 method for class 'summary.quad'
print(x, ..., dp=3)
```

Arguments

object	A quadrature scheme.
...	Ignored.
checkdup	Logical value indicating whether to test for duplicated points.
dp	Number of significant digits to print.
x	Object of class "summary.quad" returned by summary.quad.

Details

This is a method for the generic summary for the class "quad". An object of class "quad" describes a quadrature scheme, used to fit a point process model. See quad.object) for details of this class.

summary.quad extracts information about the quadrature scheme, and print.summary.quad prints this information in a comprehensible format.

In normal usage, print.summary.quad is invoked implicitly when the user calls summary.quad without assigning its value to anything. See the examples.

Value

summary.quad returns an object of class "summary.quad", while print.summary.quad returns NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
# make a quadrature scheme
Q <- quadscheme(rpoispp(42))
# summarize it
summary(Q)
# save the summary
s <- summary(Q)
# print it
print(s)
s
# extract total quadrature weight
s$w$all$sum
```

---

summary.solist *Summary of a List of Spatial Objects*

Description

Prints a useful summary of each entry in a list of two-dimensional spatial objects.

Usage

```
## S3 method for class 'solist'
summary(object, ...)
```

Arguments

object	An object of class "solist".
...	Ignored.

Details

This is a method for the generic function summary.

An object of the class "solist" is effectively a list of two-dimensional spatial datasets. See solist.

This function extracts a useful summary of each of the datasets.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

solist, summary, plot.solist

Examples

```
x <- solist(cells, japanesepines, redwood)
summary(x)
```

---

summary.splitppp *Summary of a Split Point Pattern*

Description

Prints a useful summary of a split point pattern.

Usage

```
 ## S3 method for class 'splitppp'
summary(object, ...)
```

Arguments

object	Split point pattern (object of class "splitppp", effectively a list of point patterns, usually created by split.ppp).
...	Ignored.

Details

This is a method for the generic function summary.

An object of the class "splitppp" is effectively a list of point patterns (objects of class "ppp") representing different sub-patterns of an original point pattern.

This function extracts a useful summary of each of the sub-patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

summary, split, split.ppp

Examples

```
data(amacrine)      # multitype point pattern
summary(split(amacrine))
```

---

sumouter *Compute Quadratic Forms*

Description

Calculates certain quadratic forms of matrices.

Usage

```
sumouter(x, w=NULL, y=x)
quadform(x, v)
bilinearform(x, v, y)
```

Arguments

x,y	A matrix, whose rows are the vectors in the quadratic form.
w	Optional vector of weights
v	Matrix determining the quadratic form

Details

The matrices x and y will be interpreted as collections of row vectors. They must have the same number of rows.

The command sumouter computes the sum of the outer products of corresponding row vectors, weighted by the entries of w:

$$M = \sum_i w_i x_i y_i^\top$$

where the sum is over all rows of x (after removing any rows containing NA or other non-finite values). If w is missing, the weights will be taken as 1. The result is a $p \times q$ matrix where p = ncol(x) and q = ncol(y).

The command quadform evaluates the quadratic form, defined by the matrix v, for each of the row vectors of x:

$$y_i = x_i V x_i^\top$$

The result y is a numeric vector of length n where n = nrow(x). If x[i,] contains NA or other non-finite values, then y[i] = NA.

The command bilinearform evaluates the more general bilinear form defined by the matrix v. Here x and y must be matrices of the same dimensions. For each row vector of x and corresponding row vector of y, the bilinear form is

$$z_i = x_i V y_i^\top$$

The result z is a numeric vector of length n where n = nrow(x). If x[i,] or y[i,] contains NA or other non-finite values, then z[i] = NA.

Value

A vector or matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
x <- matrix(1:12, 4, 3)
dimnames(x) <- list(c("Wilma", "Fred", "Barney", "Betty"), letters[1:3])
x

sumouter(x)

w <- 4:1
sumouter(x, w)
v <- matrix(1, 3, 3)
quadform(x, v)

# should be the same as quadform(x, v)
bilinearform(x, v, x)

# See what happens with NA's
x[3,2] <- NA
sumouter(x, w)
quadform(x, v)
```

---

superimpose *Superimpose Several Geometric Patterns*

Description

Superimpose any number of point patterns or line segment patterns.

Usage

```
 superimpose(...)

 ## S3 method for class 'ppp'
superimpose(..., W=NULL, check=TRUE)

 ## S3 method for class 'psp'
superimpose(..., W=NULL, check=TRUE)

 ## S3 method for class 'splitppp'
superimpose(..., W=NULL, check=TRUE)

 ## S3 method for class 'ppplist'
superimpose(..., W=NULL, check=TRUE)
```

```
  ## Default S3 method:
superimpose(...)
```

Arguments

...	Any number of arguments, each of which represents either a point pattern or a line segment pattern or a list of point patterns.
W	Optional. Data determining the window for the resulting pattern. Either a window (object of class "owin", or something acceptable to as.owin), or a function which returns a window, or one of the strings "convex", "rectangle", "bbox" or "none".
check	Logical value (passed to ppp or psp as appropriate) determining whether to check the geometrical validity of the resulting pattern.

Details

This function is used to superimpose several geometric patterns of the same kind, producing a single pattern of the same kind.

The function superimpose is generic, with methods for the class ppp of point patterns, the class psp of line segment patterns, and a default method. There is also a method for lpp, described separately in superimpose.lpp.

The dispatch to a method is initially determined by the class of the *first* argument in ....

- default: If the first argument is *not* an object of class ppp or psp, then the default method superimpose.default is executed. This checks the class of all arguments, and dispatches to the appropriate method. Arguments of class ppplist can be handled.
- ppp: If the first ... argument is an object of class ppp then the method superimpose.ppp is executed. All arguments in ... must be either ppp objects or lists with components x and y. The result will be an object of class ppp.
- psp: If the first ... argument is an object of class psp then the psp method is dispatched and all ... arguments must be psp objects. The result is a psp object.

The patterns are *not* required to have the same window of observation.

The window for the superimposed pattern is controlled by the argument W.

- If W is a window (object of class "W" or something acceptable to as.owin) then this determines the window for the superimposed pattern.
- If W is NULL, or the character string "none", then windows are extracted from the geometric patterns, as follows. For superimpose.psp, all arguments ... are line segment patterns (objects of class "psp"); their observation windows are extracted; the union of these windows is computed; and this union is taken to be the window for the superimposed pattern. For superimpose.ppp and superimpose.default, the arguments ... are inspected, and any arguments which are point patterns (objects of class "ppp") are selected; their observation windows are extracted, and the union of these windows is taken to be the window for the superimposed point pattern. For superimpose.default if none of the arguments is of class "ppp" then no window is computed and the result of superimpose is a list(x,y).
- If W is one of the strings "convex", "rectangle" or "bbox" then a window for the superimposed pattern is computed from the coordinates of the points or the line segments as follows.

  "bbox": the bounding box of the points or line segments (see bounding.box.xy);

  "convex": the Ripley-Rasson estimator of a convex window (see ripras);

"rectangle": the Ripley-Rasson estimator of a rectangular window (using ripras with argument shape="rectangle").

- If W is a function, then this function is used to compute a window for the superimposed pattern from the coordinates of the points or the line segments. The function should accept input of the form list(x,y) and is expected to return an object of class "owin". Examples of such functions are ripras and bounding.box.xy.

The arguments ... may be *marked* patterns. The marks of each component pattern must have the same format. Numeric and character marks may be "mixed". If there is such mixing then the numeric marks are coerced to character in the combining process. If the mark structures are all data frames, then these data frames must have the same number of columns and identical column names.

If the arguments ... are given in the form name=value, then the names will be used as an extra column of marks attached to the elements of the corresponding patterns.

Value

For superimpose.ppp, a point pattern (object of class "ppp"). For superimpose.default, either a point pattern (object of class "ppp") or a list(x,y). For superimpose.psp, a line segment pattern (object of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

superimpose.lpp, concatxy, quadscheme.

Examples

```
# superimposing point patterns
p1  <- runifrect(30)
p2  <- runifrect(42)
s1  <- superimpose(p1,p2) # Unmarked pattern.
p3  <- list(x=rnorm(20),y=rnorm(20))
s2  <- superimpose(p3,p2,p1) # Default method gets called.
s2a <- superimpose(p1,p2,p3) # Same as s2 except for order of points.
s3  <- superimpose(clyde=p1,irving=p2) # Marked pattern; marks a factor
                                       # with levels "clyde" and "irving";
                                       # warning given.
marks(p1) <- factor(sample(LETTERS[1:3],30,TRUE))
marks(p2) <- factor(sample(LETTERS[1:3],42,TRUE))
s5  <- superimpose(clyde=p1,irving=p2) # Marked pattern with extra column
marks(p2) <- data.frame(a=marks(p2),b=runif(42))
s6  <- try(superimpose(p1,p2)) # Gives an error.
marks(p1) <- data.frame(a=marks(p1),b=1:30)
s7  <- superimpose(p1,p2) # O.K.

# how to make a 2-type point pattern with types "a" and "b"
u <- superimpose(a = rpoispp(10), b = rpoispp(20))

# how to make a 2-type point pattern with types 1 and 2
u <- superimpose("1" = rpoispp(10), "2" = rpoispp(20))
```

```
# superimposing line segment patterns
X <- rpoisline(10)
Y <- as.psp(matrix(runif(40), 10, 4), window=owin())
Z <- superimpose(X, Y)

# being unreasonable
## Not run:
 if(FALSE) {
  crud <- try(superimpose(p1,p2,X,Y)) # Gives an error, of course!
 }

## End(Not run)
```

superimpose.lpp *Superimpose Several Point Patterns on Linear Network*

Description

Superimpose any number of point patterns on the same linear network.

Usage

```
  ## S3 method for class 'lpp'
superimpose(..., L=NULL)
```

Arguments

...	Any number of arguments, each of which represents a point pattern on the same linear network. Each argument can be either an object of class "lpp", giving both the spatial coordinates of the points and the linear network, or a list(x,y) or list(x,y,seg,tp) giving just the spatial coordinates of the points.
L	Optional. The linear network. An object of class "linnet". This argument is required if none of the other arguments is of class "lpp".

Details

This function is used to superimpose several point patterns on the same linear network. It is a method for the generic function superimpose.

Each of the arguments ... can be either a point pattern on a linear network (object of class "lpp" giving both the spatial coordinates of the points and the linear network), or a list(x,y) or list(x,y,seg,tp) giving just the spatial coordinates of the points. These arguments must represent point patterns on the *same* linear network.

The argument L is an alternative way to specify the linear network, and is required if none of the arguments ... is an object of class "lpp".

The arguments ... may be *marked* patterns. The marks of each component pattern must have the same format. Numeric and character marks may be “mixed”. If there is such mixing then the numeric marks are coerced to character in the combining process. If the mark structures are all data frames, then these data frames must have the same number of columns and identical column names.

If the arguments ... are given in the form name=value, then the names will be used as an extra column of marks attached to the elements of the corresponding patterns.

Value

An object of class "lpp" representing the combined point pattern on the linear network.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

Ege Rubak <rubak@math.aau.dk>

and Greg McSwiggan.

See Also

superimpose

Examples

```
X <- rpoislpp(5, simplenet)
Y <- rpoislpp(10, simplenet)
superimpose(X,Y) # not marked
superimpose(A=X, B=Y) # multitype with types A and B
```

---

symbolmap *Graphics Symbol Map*

Description

Create a graphics symbol map that associates data values with graphical symbols.

Usage

```
symbolmap(..., range = NULL, inputs = NULL)
```

Arguments

...	Named arguments specifying the graphical parameters. See Details.
range	Optional. Range of numbers that are mapped. A numeric vector of length 2 giving the minimum and maximum values that will be mapped. Incompatible with inputs.
inputs	Optional. A vector containing all the data values that will be mapped to symbols. Incompatible with range.

Details

A graphical symbol map is an association between data values and graphical symbols. The command symbolmap creates an object of class "symbolmap" that represents a graphical symbol map.

Once a symbol map has been created, it can be applied to any suitable data to generate a plot of those data. This makes it easy to ensure that the *same* symbol map is used in two different plots. The symbol map can be plotted as a legend to the plots, and can also be plotted in its own right.

The possible values of data that will be mapped are specified by range or inputs.

- if range is given, it should be a numeric vector of length 2 giving the minimum and maximum values of the range of numbers that will be mapped. These limits must be finite.
- if inputs is given, it should be a vector of any atomic type (e.g. numeric, character, logical, factor). This vector contains all the possible data values that will be mapped.
- If neither range nor inputs is given, it is assumed that the possible values are real numbers.

The association of data values with graphical symbols is specified by the other arguments ... which are given in name=value form. These arguments specify the kinds of symbols that will be used, the sizes of the symbols, and graphics parameters for drawing the symbols.

Each graphics parameter can be either a single value, for example shape="circles", or a function(x) which determines the value of the graphics parameter as a function of the data x, for example shape=function(x) ifelse(x > 0, "circles", "squares"). Colourmaps (see colourmap) are also acceptable because they are functions.

Currently recognised graphics parameters, and their allowed values, are:

shape The shape of the symbol: currently either "circles", "squares", "arrows" or NA. This parameter takes precedence over pch.

size The size of the symbol: a positive number or zero.

pch Graphics character code: a positive integer, or a single character. See par.

cex Graphics character expansion factor.

cols Colour of plotting characters.

fg,bg Colour of foreground (or symbol border) and background (or symbol interior).

col,lwd,lty Colour, width and style of lines.

etch Logical. If TRUE, each symbol is surrounded by a border drawn in the opposite colour, which improves its visibility against the background. Default is FALSE.

direction,headlength,headangle,arrowtype Numeric parameters of arrow symbols, applicable when shape="arrows". Here direction is the direction of the arrow in degrees anticlockwise from the x axis; headlength is the length of the head of the arrow in coordinate units; headangle is the angle subtended by the point of the arrow; and arrowtype is an integer code specifying which ends of the shaft have arrowheads attached (0 means no arrowheads, 1 is an arrowhead at the start of the shaft, 2 is an arrowhead at the end of the shaft, and 3 is arrowheads at both ends).

A vector of colour values is also acceptable for the arguments col,cols,fg,bg if range is specified.

Value

An object of class "symbolmap".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

plot.symbolmap to plot the symbol map itself.

invoke.symbolmap to apply the symbol map to some data and plot the resulting symbols.

update.symbolmap to change the symbol map.

Examples

```
g <- symbolmap(inputs=letters[1:10], pch=11:20)

g1 <- symbolmap(range=c(0,100), size=function(x) x/50)

g2 <- symbolmap(shape=function(x) ifelse(x > 0, "circles", "squares"),
                  size=function(x) sqrt(ifelse(x > 0, x/pi, -x)),
                  bg = function(x) ifelse(abs(x) < 1, "red", "black"))

colmap <- colourmap(topo.colors(20), range=c(0,10))
g3 <- symbolmap(pch=21, bg=colmap, range=c(0,10))
plot(g3)
```

`tess` *Create a Tessellation*

Description

Creates an object of class `"tess"` representing a tessellation of a spatial region.

Usage

```
tess(..., xgrid = NULL, ygrid = NULL, tiles = NULL, image = NULL,
          window=NULL, marks=NULL, keepempty=FALSE, unitname=NULL, check=TRUE)
```

Arguments

`...`	Ignored.
`xgrid,ygrid`	Cartesian coordinates of vertical and horizontal lines determining a grid of rectangles. Incompatible with other arguments.
`tiles`	List of tiles in the tessellation. A list, each of whose elements is a window (object of class `"owin"`). Incompatible with other arguments.
`image`	Pixel image which specifies the tessellation. Incompatible with other arguments.
`window`	Optional. The spatial region which is tessellated (i.e. the union of all the tiles). An object of class `"owin"`.
`marks`	Optional vector or data frame of marks associated with the tiles.
`keepempty`	Logical flag indicating whether empty tiles should be retained or deleted.
`unitname`	Optional. Name of unit of length. Either a single character string, or a vector of two character strings giving the singular and plural forms, respectively. If this argument is missing or `NULL`, information about the unitname will be extracted from the other arguments. If this argument is given, it overrides any other information about the unitname.
`check`	Logical value indicating whether to check the validity of the input data. It is strongly recommended to use the default value `check=TRUE`.

Details

A tessellation is a collection of disjoint spatial regions (called *tiles*) that fit together to form a larger spatial region. This command creates an object of class "tess" that represents a tessellation.

Three types of tessellation are supported:

rectangular: tiles are rectangles, with sides parallel to the x and y axes. They may or may not have equal size and shape. The arguments xgrid and ygrid determine the positions of the vertical and horizontal grid lines, respectively. (See quadrats for another way to do this.)

tile list: tiles are arbitrary spatial regions. The argument tiles is a list of these tiles, which are objects of class "owin".

pixel image: Tiles are subsets of a fine grid of pixels. The argument image is a pixel image (object of class "im") with factor values. Each level of the factor represents a different tile of the tessellation. The pixels that have a particular value of the factor constitute a tile.

The optional argument window specifies the spatial region formed by the union of all the tiles. In other words it specifies the spatial region that is divided into tiles by the tessellation. If this argument is missing or NULL, it will be determined by computing the set union of all the tiles. This is a time-consuming computation. For efficiency it is advisable to specify the window. Note that the validity of the window will not be checked.

Empty tiles may occur, either because one of the entries in the list tiles is an empty window, or because one of the levels of the factor-valued pixel image image does not occur in the pixel data. When keepempty=TRUE, empty tiles are permitted. When keepempty=FALSE (the default), tiles are not allowed to be empty, and any empty tiles will be removed from the tessellation.

There are methods for print, plot, [and [<- for tessellations. Use tiles to extract the list of tiles in a tessellation, tilenames to extract the names of the tiles, and tile.areas to compute their areas.

The tiles may have marks, which can be extracted by marks.tess and changed by marks<-.tess.

Tessellations can be used to classify the points of a point pattern, in split.ppp, cut.ppp and by.ppp.

To construct particular tessellations, see quadrats, hextess, dirichlet, delaunay and rpoislinetess.

Value

An object of class "tess" representing the tessellation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

marks.tess, plot.tess, [.tess, as.tess, tiles, intersect.tess, split.ppp, cut.ppp, by.ppp, bdist.tiles, tile.areas.

To construct particular tessellations, see quadrats, hextess, dirichlet, delaunay and rpoislinetess.

To divide space into pieces containing equal amounts of stuff, use quantess.

Examples

```
A <- tess(xgrid=0:4,ygrid=0:4)
A
B <- A[c(1, 2, 5, 7, 9)]
B
v <- as.im(function(x,y){factor(round(5 * (x^2 + y^2)))}, W=owin())
levels(v) <- letters[seq(length(levels(v)))]
E <- tess(image=v)
E
```

`test.crossing.psp` *Check Whether Segments Cross*

Description

Determine whether there is a crossing (intersection) between each pair of line segments.

Usage

```
test.crossing.psp(A, B)
test.selfcrossing.psp(A)
```

Arguments

`A,B`	Line segment patterns (objects of class "psp").

Details

These functions decide whether the given line segments intersect each other.

If `A` and `B` are two spatial patterns of line segments, `test.crossing.psp(A, B)` returns a logical matrix in which the entry on row `i`, column `j` is equal to `TRUE` if segment `A[i]` has an intersection with segment `B[j]`.

If `A` is a pattern of line segments, `test.selfcross.psp(A)` returns a symmetric logical matrix in which the entry on row `i`, column `j` is equal to `TRUE` if segment `A[i]` has an intersection with segment `A[j]`.

Value

A logical matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

psp

Examples

```
B <- edges(letterR)
A <- rpoisline(5, Frame(B))
MA <- test.selfcrossing.psp(A)
MAB <- test.crossing.psp(A, B)
```

---

text.ppp *Add Text Labels to Spatial Pattern*

Description

Plots a text label at the location of each point in a spatial point pattern, or each object in a spatial pattern of objects.

Usage

```
## S3 method for class 'ppp'
text(x, ...)

## S3 method for class 'lpp'
text(x, ...)

## S3 method for class 'psp'
text(x, ...)
```

Arguments

x	A spatial point pattern (object of class "ppp"), a point pattern on a linear network (class "lpp") or a spatial pattern of line segments (class "psp").
...	Additional arguments passed to `text.default`.

Details

These functions are methods for the generic `text`. A text label is added to the existing plot, at the location of each point in the point pattern `x`, or near the location of the midpoint of each segment in the segment pattern `x`.

Additional arguments `...` are passed to `text.default` and may be used to control the placement of the labels relative to the point locations, and the size and colour of the labels.

By default, the labels are the serial numbers 1 to `n`, where `n` is the number of points or segments in `x`. This can be changed by specifying the argument `labels`, which should be a vector of length `n`.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`text.default`

Examples

```
plot(cells)
text(cells, pos=2)

plot(Frame(cells))
text(cells, cex=1.5)

S <- as.psp(simplenet)
plot(S)
text(S)
```

---

texturemap *Texture Map*

Description

Create a map that associates data values with graphical textures.

Usage

```
texturemap(inputs, textures, ...)
```

Arguments

`inputs`	A vector containing all the data values that will be mapped to textures.
`textures`	Optional. A vector of integer codes specifying the textures to which the `inputs` will be mapped.
`...`	Other graphics parameters such as `col`, `lwd`, `lty`.

Details

A texture map is an association between data values and graphical textures. The command `texturemap` creates an object of class `"texturemap"` that represents a texture map.

Once a texture map has been created, it can be applied to any suitable data to generate a texture plot of those data using `textureplot`. This makes it easy to ensure that the *same* texture map is used in two different plots. The texture map can also be plotted in its own right.

The argument `inputs` should be a vector containing all the possible data values (such as the levels of a factor) that are to be mapped.

The `textures` should be integer values between 1 and 8, representing the eight possible textures described in the help for `add.texture`. The default is `textures = 1:n` where `n` is the length of `inputs`.

Value

An object of class `"texturemap"` representing the texture map.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

textureplot

Examples

```
texturemap(letters[1:4], 2:5, col=1:4, lwd=2)
```

---

textureplot *Plot Image or Tessellation Using Texture Fill*

Description

For a factor-valued pixel image, this command plots each level of the factor using a different texture. For a tessellation, each tile is plotted using a different texture.

Usage

```
textureplot(x, ...,
            main, add=FALSE, clipwin=NULL, do.plot = TRUE,
            border=NULL, col = NULL, lwd = NULL, lty = NULL, spacing = NULL,
            textures=1:8,
            legend=TRUE,
            leg.side=c("right", "left", "bottom", "top"),
            legsep=0.1, legwid=0.2)
```

Arguments

`x`	A tessellation (object of class "tess" or something acceptable to `as.tess`) with at most 8 tiles, or a pixel image (object of class "im" or something acceptable to `as.im`) whose pixel values are a `factor` with at most 8 levels.
`...`	Other arguments passed to `add.texture`.
`main`	Character string giving a main title for the plot.
`add`	Logical value indicating whether to draw on the current plot (add=TRUE) or to initialise a new plot (add=FALSE).
`clipwin`	Optional. A window (object of class "owin"). Only this subset of the image will be displayed.
`do.plot`	Logical. Whether to actually do the plot.
`border`	Colour for drawing the boundaries between the different regions. The default (border=NULL) means to use par("fg"). Use border=NA to omit borders.
`col`	Numeric value or vector giving the colour or colours in which the textures should be plotted.
`lwd`	Numeric value or vector giving the line width or widths to be used.
`lty`	Numeric value or vector giving the line type or types to be used.

spacing	Numeric value or vector giving the spacing parameter for the textures.
textures	Textures to be used for each level. Either a texture map (object of class "texturemap") or a vector of integer codes (to be interpreted by add.texture).
legend	Logical. Whether to display an explanatory legend.
leg.side	Position of legend relative to main plot.
legsep	Separation between legend and main plot, as a fraction of the shortest side length of the main plot.
legwid	Width (if vertical) or height (if horizontal) of the legend as a fraction of the shortest side length of the main plot.

Details

If x is a tessellation, then each tile of the tessellation is plotted and filled with a texture using add.texture.

If x is a factor-valued pixel image, then for each level of the factor, the algorithm finds the region where the image takes this value, and fills the region with a texture using add.texture.

Value

(Invisible) A texture map (object of class "texturemap") associating a texture with each level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

im, plot.im, add.texture.

Examples

```
nd <- if(interactive()) 128 else 32
Z <- setcov(owin(), dimyx=nd)
Zcut <- cut(Z, 3, labels=c("Lo", "Med", "Hi"))
textureplot(Zcut)
textureplot(dirichlet(runifpoint(6)))
```

---

thinNetwork *Remove Vertices or Segments from a Linear Network*

Description

Delete some vertices and/or segments from a linear network or related object.

Usage

```
thinNetwork(X, retainvertices, retainedges)
```

Arguments

- `X` — A linear network (object of class "linnet"), or a point pattern on a linear network (object of class "lpp").
- `retainvertices` — Optional. Subset index specifying which vertices should be retained (not deleted).
- `retainedges` — Optional. Subset index specifying which edges (segments) should be retained (not deleted).

Details

This function deletes some of the vertices and edges (segments) in the linear network.

The arguments `retainvertices` and `retainedges` can be any kind of subset index: a vector of positive integers specifying which vertices/edges should be retained; a vector of negative integers specifying which vertices/edges should be deleted; or a logical vector specifying whether each vertex/edge should be retained (`TRUE`) or deleted (`FALSE`).

Vertices are indexed in the same sequence as in `vertices(as.linnet(X))`. Segments are indexed in the same sequence as in `as.psp(as.linnet(X))`.

The argument `retainedges` has higher precedence than `retainvertices` in the sense that:

- If `retainedges` is given, then any vertex which is an endpoint of a retained edge will also be retained.
- If `retainvertices` is given and `retainedges` is **missing**, then any segment joining two retained vertices will also be retained.
- Thus, when both `retainvertices` and `retainedges` are given, it is possible that more vertices will be retained than those specified by `retainvertices`.

After the network has been altered, other consequential changes will occur, including renumbering of the segments and vertices. If `X` is a point pattern on a linear network, then data points will be deleted if they lie on a deleted edge.

Value

An object of the same kind as `X`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Suman Rakshit.

See Also

`linnet` to make a network;

`connected.linnet` to extract connected components.

Examples

```
L <- simplenet
plot(L, main="thinNetwork(L, retainedges=c(-3, -5))")
text(midpoints.psp(as.psp(L)), labels=1:nsegments(L), pos=3)
Lsub <- thinNetwork(L, retainedges=c(-3, -5))
plot(Lsub, add=TRUE, col="blue", lwd=2)
```

thomas.estK *Fit the Thomas Point Process by Minimum Contrast*

Description

Fits the Thomas point process to a point pattern dataset by the Method of Minimum Contrast using the K function.

Usage

```
thomas.estK(X, startpar=c(kappa=1,scale=1), lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...)
```

Arguments

X	Data to which the Thomas model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the Thomas process.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to `optim` to control the optimisation algorithm. See Details.

Details

This algorithm fits the Thomas point process model to a point pattern dataset by the Method of Minimum Contrast, using the K function.

The argument `X` can be either

a point pattern: An object of class `"ppp"` representing a point pattern dataset. The K function of the point pattern will be computed using `Kest`, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class `"fv"` containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the K function, and this object should have been obtained by a call to `Kest` or one of its relatives.

The algorithm fits the Thomas point process to `X`, by finding the parameters of the Thomas model which give the closest match between the theoretical K function of the Thomas process and the observed K function. For a more detailed explanation of the Method of Minimum Contrast, see `mincontrast`.

The Thomas point process is described in Møller and Waagepetersen (2003, pp. 61–62). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent are independent and isotropically Normally distributed around the parent point with standard deviation σ which is equal to the parameter `scale`. The named vector of stating values can use either `sigma2` (σ^2) or `scale` as the name of the second component, but the latter is recommended for consistency with other cluster models.

The theoretical K-function of the Thomas process is

$$K(r) = \pi r^2 + \frac{1}{\kappa}(1 - \exp(-\frac{r^2}{4\sigma^2})).$$

The theoretical intensity of the Thomas process is $\lambda = \kappa\mu$.

In this algorithm, the Method of Minimum Contrast is first used to find optimal values of the parameters κ and σ^2. Then the remaining parameter μ is inferred from the estimated intensity λ.

If the argument `lambda` is provided, then this is used as the value of λ. Otherwise, if `X` is a point pattern, then λ will be estimated from `X`. If `X` is a summary statistic and `lambda` is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as `NA`.

The remaining arguments `rmin,rmax,q,p` control the method of minimum contrast; see `mincontrast`.

The Thomas process can be simulated, using `rThomas`.

Homogeneous or inhomogeneous Thomas process models can also be fitted using the function `kppm`.

The optimisation algorithm can be controlled through the additional arguments `"..."` which are passed to the optimisation function `optim`. For example, to constrain the parameter values to a certain range, use the argument `method="L-BFGS-B"` to select an optimisation algorithm that respects box constraints, and use the arguments `lower` and `upper` to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class `"minconfit"`. There are methods for printing and plotting this object. It contains the following main components:

`par`	Vector of fitted parameter values.
`fit`	Function value table (object of class `"fv"`) containing the observed values of the summary statistic (`observed`) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Rasmus Waagepetersen <rw@math.auc.dk> Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu

References

Diggle, P. J., Besag, J. and Gleaves, J. T. (1976) Statistical analysis of spatial point patterns by means of distance methods. *Biometrics* **32** 659–667.

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Thomas, M. (1949) A generalisation of Poisson's binomial limit for use in ecology. *Biometrika* **36**, 18–25.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

`kppm`, `lgcp.estK`, `matclust.estK`, `mincontrast`, `Kest`, `rThomas` to simulate the fitted model.

Examples

```
data(redwood)
u <- thomas.estK(redwood, c(kappa=10, scale=0.1))
u
plot(u)
```

---

thomas.estpcf *Fit the Thomas Point Process by Minimum Contrast*

Description

Fits the Thomas point process to a point pattern dataset by the Method of Minimum Contrast using the pair correlation function.

Usage

```
thomas.estpcf(X, startpar=c(kappa=1,scale=1), lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ..., pcfargs=list())
```

Arguments

X	Data to which the Thomas model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the Thomas process.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.
pcfargs	Optional list containing arguments passed to pcf.ppp to control the smoothing in the estimation of the pair correlation function.

Details

This algorithm fits the Thomas point process model to a point pattern dataset by the Method of Minimum Contrast, using the pair correlation function pcf.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The pair correlation function of the point pattern will be computed using pcf, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the pair correlation function, and this object should have been obtained by a call to pcf or one of its relatives.

The algorithm fits the Thomas point process to X, by finding the parameters of the Thomas model which give the closest match between the theoretical pair correlation function of the Thomas process and the observed pair correlation function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The Thomas point process is described in Møller and Waagepetersen (2003, pp. 61–62). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent are independent and isotropically Normally distributed around the parent point with standard deviation σ which is equal to the parameter `scale`. The named vector of stating values can use either `sigma2` (σ^2) or `scale` as the name of the second component, but the latter is recommended for consistency with other cluster models.

The theoretical pair correlation function of the Thomas process is

$$g(r) = 1 + \frac{1}{4\pi\kappa\sigma^2}\exp(-\frac{r^2}{4\sigma^2})).$$

The theoretical intensity of the Thomas process is $\lambda = \kappa\mu$.

In this algorithm, the Method of Minimum Contrast is first used to find optimal values of the parameters κ and σ^2. Then the remaining parameter μ is inferred from the estimated intensity λ.

If the argument `lambda` is provided, then this is used as the value of λ. Otherwise, if `X` is a point pattern, then λ will be estimated from `X`. If `X` is a summary statistic and `lambda` is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as `NA`.

The remaining arguments `rmin,rmax,q,p` control the method of minimum contrast; see `mincontrast`.

The Thomas process can be simulated, using `rThomas`.

Homogeneous or inhomogeneous Thomas process models can also be fitted using the function `kppm`.

The optimisation algorithm can be controlled through the additional arguments `"..."` which are passed to the optimisation function `optim`. For example, to constrain the parameter values to a certain range, use the argument `method="L-BFGS-B"` to select an optimisation algorithm that respects box constraints, and use the arguments `lower` and `upper` to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class `"minconfit"`. There are methods for printing and plotting this object. It contains the following main components:

`par`	Vector of fitted parameter values.
`fit`	Function value table (object of class `"fv"`) containing the observed values of the summary statistic (`observed`) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Diggle, P. J., Besag, J. and Gleaves, J. T. (1976) Statistical analysis of spatial point patterns by means of distance methods. *Biometrics* **32** 659–667.

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Thomas, M. (1949) A generalisation of Poisson's binomial limit for use in ecology. *Biometrika* **36**, 18–25.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

thomas.estK mincontrast, pcf, rThomas to simulate the fitted model.

Examples

```
data(redwood)
u <- thomas.estpcf(redwood, c(kappa=10, scale=0.1))
u
plot(u, legendpos="topright")
u2 <- thomas.estpcf(redwood, c(kappa=10, scale=0.1),
      pcfargs=list(stoyan=0.12))
```

---

tile.areas *Compute Areas of Tiles in a Tessellation*

Description

Computes the area of each tile in a tessellation.

Usage

```
tile.areas(x)
```

Arguments

x	A tessellation (object of class "tess").

Details

A tessellation is a collection of disjoint spatial regions (called *tiles*) that fit together to form a larger spatial region. See tess.

This command computes the area of each of the tiles that make up the tessellation x. The result is a numeric vector in the same order as the tiles would be listed by tiles(x).

Value

A numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

tess, tiles, tilenames, tiles.empty

Examples

```
A <- tess(xgrid=0:2,ygrid=0:2)
tile.areas(A)
v <- as.im(function(x,y){factor(round(x^2 + y^2))}, W=owin())
E <- tess(image=v)
tile.areas(E)
```

tile.lengths *Compute Lengths of Tiles in a Tessellation on a Network*

Description

Computes the length of each tile in a tessellation on a linear network.

Usage

```
tile.lengths(x)
```

Arguments

x	A tessellation on a linear network (object of class "lintess").

Details

A tessellation on a linear network L is a partition of the network into non-overlapping pieces (tiles). Each tile consists of one or more line segments which are subsets of the line segments making up the network. A tile can consist of several disjoint pieces.

This command computes the length of each of the tiles that make up the tessellation x. The result is a numeric vector.

Value

A numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

lintess

Examples

```
X <- runiflpp(5, simplenet)
A <- lineardirichlet(X)
plot(A)
tile.lengths(A)
```

tileindex	*Determine Which Tile Contains Each Given Point*

Description

Given a tessellation and a list of spatial points, determine which tile of the tessellation contains each of the given points.

Usage

```
tileindex(x, y, Z)
```

Arguments

`x,y`	Spatial coordinates. Numeric vectors of equal length.
`Z`	A tessellation (object of class `"tess"`).

Details

This function determines which tile of the tessellation `Z` contains each of the spatial points with coordinates `(x[i],y[i])`.

The result is a factor, of the same length as `x` and `y`, indicating which tile contains each point. The levels of the factor are the names of the tiles of `Z`. Values are `NA` if the corresponding point lies outside the tessellation.

Value

A factor, of the same length as `x` and `y`, whose levels are the names of the tiles of `Z`.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`, Rolf Turner `<r.turner@auckland.ac.nz>` and Ege Rubak `<rubak@math.aau.dk>`

See Also

`cut.ppp` and `split.ppp` to divide up the points of a point pattern according to a tessellation.

`as.function.tess` to create a function whose value is the tile index.

Examples

```
X <- runifpoint(7)
V <- dirichlet(X)
tileindex(0.1, 0.4, V)
```

tilenames *Names of Tiles in a Tessellation*

Description

Extract or Change the Names of the Tiles in a Tessellation.

Usage

```
tilenames(x)
tilenames(x) <- value
```

Arguments

x	A tessellation (object of class "tess").
value	Character vector giving new names for the tiles.

Details

These functions extract or change the names of the tiles that make up the tessellation x.

If the tessellation is a regular grid, the tile names cannot be changed.

Value

tilenames returns a character vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

tess, tiles

Examples

```
D <- dirichlet(runifpoint(10))
tilenames(D)
tilenames(D) <- paste("Cell", 1:10)
```

tiles	*Extract List of Tiles in a Tessellation*

Description

Extracts a list of the tiles that make up a tessellation.

Usage

```
tiles(x)
```

Arguments

x	A tessellation (object of class "tess").

Details

A tessellation is a collection of disjoint spatial regions (called *tiles*) that fit together to form a larger spatial region. See tess.

The tiles that make up the tessellation x are returned in a list.

Value

A list of windows (objects of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

tess, tilenames, tile.areas, tiles.empty

Examples

```
A <- tess(xgrid=0:2,ygrid=0:2)
tiles(A)
v <- as.im(function(x,y){factor(round(x^2 + y^2))}, W=owin())
E <- tess(image=v)
tiles(E)
```

`tiles.empty` *Check For Empty Tiles in a Tessellation*

Description

Checks whether each tile in a tessellation is empty or non-empty.

Usage

```
tiles.empty(x)
```

Arguments

`x`	A tessellation (object of class `"tess"`).

Details

A tessellation is a collection of disjoint spatial regions (called *tiles*) that fit together to form a larger spatial region. See `tess`.

It is possible for some tiles of a tessellation to be empty. For example, this can happen when the tessellation `x` is obtained by restricting another tessellation `y` to a smaller spatial domain `w`.

The function `tiles.empty` checks whether each tile is empty or non-empty. The result is a logical vector, with entries equal to `TRUE` when the corresponding tile is empty. Results are given in the same order as the tiles would be listed by `tiles(x)`.

Value

A logical vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`tess`, `tiles`, `tilenames`, `tile.areas`

Examples

```
A <- tess(xgrid=0:2,ygrid=0:2)
tiles.empty(A)
v <- as.im(function(x,y){factor(round(x^2 + y^2))}, W=owin())
E <- tess(image=v)
tiles.empty(E)
```

`timed` *Record the Computation Time*

Description

Saves the result of a calculation as an object of class `"timed"` which includes information about the time taken to compute the result. The computation time is printed when the object is printed.

Usage

```
timed(x, ..., starttime = NULL, timetaken = NULL)
```

Arguments

`x`	An expression to be evaluated, or an object that has already been evaluated.
`starttime`	The time at which the computation is defined to have started. The default is the current time. Ignored if `timetaken` is given.
`timetaken`	The length of time taken to perform the computation. The default is the time taken to evaluate `x`.
`...`	Ignored.

Details

This is a simple mechanism for recording how long it takes to perform complicated calculations (usually for the purposes of reporting in a publication).

If `x` is an expression to be evaluated, `timed(x)` evaluates the expression and measures the time taken to evaluate it. The result is saved as an object of the class `"timed"`. Printing this object displays the computation time.

If `x` is an object which has already been computed, then the time taken to compute the object can be specified either directly by the argument `timetaken`, or indirectly by the argument `starttime`.

- `timetaken` is the duration of time taken to perform the computation. It should be the difference of two clock times returned by `proc.time`. Typically the user sets `begin <- proc.time()` before commencing the calculations, then `end <- proc.time()` after completing the calculations, and then sets `timetaken <- end - begin`.
- `starttime` is the clock time at which the computation started. It should be a value that was returned by `proc.time` at some earlier time when the calculations commenced. When `timed` is called, the computation time will be taken as the difference between the current clock time and `starttime`. Typically the user sets `begin <- proc.time()` before commencing the calculations, and when the calculations are completed, the user calls `result <- timed(result, starttime=begin)`.

If the result of evaluating `x` belongs to other S3 classes, then the result of `timed(x, ...)` also inherits these classes, and printing the object will display the appropriate information for these classes as well.

Value

An object inheriting the class `"timed"`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

timeTaken to extract the time taken.

Examples

```
timed(clarkevans(cells))

timed(Kest(cells))

answer <- timed(42, timetaken=4.1e17)
answer
```

---

timeTaken *Extract the Total Computation Time*

Description

Given an object or objects that contain timing information (reporting the amount of computer time taken to compute each object), this function extracts the timing data and evaluates the total time taken.

Usage

```
timeTaken(..., warn=TRUE)
```

Arguments

...	One or more objects of class "timed" containing timing data.
warn	Logical value indicating whether a warning should be issued if some of the arguments do not contain timing information.

Details

An object of class "timed" contains information on the amount of computer time that was taken to compute the object. See timed.

This function extracts the timing information from one or more such objects, and calculates the total time.

Value

An object inheriting the class "timed".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

timed

Examples

```
A <- timed(Kest(cells))
B <- timed(Gest(cells))
A
B
timeTaken(A,B)
```

---

transect.im — *Pixel Values Along a Transect*

Description

Extract the pixel values of a pixel image at each point along a linear transect.

Usage

```
transect.im(X, ..., from="bottomleft", to="topright",
              click=FALSE, add=FALSE)
```

Arguments

X	A pixel image (object of class "im").
...	Ignored.
from,to	Optional. Start point and end point of the transect. Pairs of (x, y) coordinates in a format acceptable to xy.coords, or keywords "bottom", "left", "top", "right", "bottomleft" etc.
click	Optional. Logical value. If TRUE, the linear transect is determined interactively by the user, who clicks two points on the current plot.
add	Logical. If click=TRUE, this argument determines whether to perform interactive tasks on the current plot (add=TRUE) or to start by plotting X (add=FALSE).

Details

The pixel values of the image X along a line segment will be extracted. The result is a function table ("fv" object) which can be plotted directly.

If click=TRUE, then the user is prompted to click two points on the plot of X. These endpoints define the transect.

Otherwise, the transect is defined by the endpoints from and to. The default is a diagonal transect from bottom left to top right of the frame.

Value

An object of class "fv" which can be plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

im

Examples

```
Z <- density(redwood)
plot(transect.im(Z))
## Not run:
 if(FALSE) {
  plot(transect.im(Z, click=TRUE))
 }

## End(Not run)
```

`transmat` *Convert Pixel Array Between Different Conventions*

Description

This function provides a simple way to convert arrays of pixel data between different display conventions.

Usage

```
transmat(m, from, to)
```

Arguments

`m`	A matrix.
`from,to`	Specifications of the spatial arrangement of the pixels. See Details.

Details

Pixel images are handled by many different software packages. In virtually all of these, the pixel values are stored in a matrix, and are accessed using the row and column indices of the matrix. However, different pieces of software use different conventions for mapping the matrix indices $[i, j]$ to the spatial coordinates (x, y).

- In the *Cartesian* convention, the first matrix index i is associated with the first Cartesian coordinate x, and j is associated with y. This convention is used in `image.default`.
- In the *European reading order* convention, a matrix is displayed in the spatial coordinate system as it would be printed in a page of text: i is effectively associated with the negative y coordinate, and j is associated with x. This convention is used in some image file formats.
- In the `spatstat` convention, i is associated with the increasing y coordinate, and j is associated with x. This is also used in some image file formats.

To convert between these conventions, use the function `transmat`. If a matrix `m` contains pixel image data that is correctly displayed by software that uses the Cartesian convention, and we wish to convert it to the European reading convention, we can type `mm <- transmat(m, from="Cartesian", to="European")`. The transformed matrix `mm` will then be correctly displayed by software that uses the European convention.

Each of the arguments `from` and `to` can be one of the names `"Cartesian"`, `"European"` or `"spatstat"` (partially matched) or it can be a list specifying another convention. For example `to=list(x="-i", y="-j")!` specifies that rows of the output matrix are expected to be displayed as vertical columns in the plot, starting at the right side of the plot, as in the traditional Chinese, Japanese and Korean writing order.

Value

Another matrix obtained by rearranging the entries of `m`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
opa <- par(mfrow=c(1,2))
# image in spatstat format
Z <- bei.extra$elev
plot(Z, main="plot.im", ribbon=FALSE)
m <- as.matrix(Z)
# convert matrix to format suitable for display by image.default
Y <- transmat(m, from="spatstat", to="Cartesian")
image(Y, asp=0.5, main="image.default", axes=FALSE)
par(opa)
```

`treebranchlabels` *Label Vertices of a Tree by Branch Membership*

Description

Given a linear network which is a tree (acyclic graph), this function assigns a label to each vertex, indicating its position in the tree.

Usage

```
treebranchlabels(L, root = 1)
```

Arguments

`L`	Linear network (object of class `"linnet"`). The network must have no loops.
`root`	Root of the tree. An integer index identifying which point in `vertices(L)` is the root of the tree.

Details

The network L should be a tree, that is, it must have no loops.

This function computes a character string label for each vertex of the network L. The vertex identified by root (that is, vertices(L)[root]) is taken as the root of the tree and is given the empty label "".

- If there are several line segments which meet at the root vertex, each of these segments is the start of a new branch of the tree; the other endpoints of these segments are assigned the labels "a", "b", "c" and so on.
- If only one segment issues from the root vertex, the other endpoint of this segment is assigned the empty label "".

A similar rule is then applied to each of the newly-labelled vertices. If the vertex labelled "a" is joined to two other unlabelled vertices, these will be labelled "aa" and "ab". The rule is applied recursively until all vertices have been labelled.

If L is not a tree, the algorithm will terminate, but the results will be nonsense.

Value

A vector of character strings, with one entry for each point in vertices(L).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

deletebranch, extractbranch, treeprune for manipulating a network using the branch labels.

linnet for creating a network.

Examples

```
# make a simple tree
m <- simplenet$m
m[8,10] <- m[10,8] <- FALSE
L <- linnet(vertices(simplenet), m)
plot(L, main="")
# compute branch labels
tb <- treebranchlabels(L, 1)
tbc <- paste0("[", tb, "]")
text(vertices(L), labels=tbc, cex=2)
```

---

treeprune *Prune Tree to Given Level*

Description

Prune a tree by removing all the branches above a given level.

Usage

```
treeprune(X, root = 1, level = 0)
```

Arguments

X	Object of class "linnet" or "lpp".
root	Index of the root vertex amongst the vertices of as.linnet(X).
level	Integer specifying the level above which the tree should be pruned.

Details

The object X must be either a linear network, or a derived object such as a point pattern on a linear network. The linear network must be an acyclic graph (i.e. must not contain any loops) so that it can be interpreted as a tree.

This function removes all vertices for which treebranchlabels gives a string more than level characters long.

Value

Object of the same kind as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

treebranchlabels for calculating the branch labels.

deletebranch for removing entire branches. extractbranch for extracting entire branches.

linnet for creating networks.

Examples

```
# make a simple tree
m <- simplenet$m
m[8,10] <- m[10,8] <- FALSE
L <- linnet(vertices(simplenet), m)
plot(L, main="")
# compute branch labels
tb <- treebranchlabels(L, 1)
tbc <- paste0("[", tb, "]")
text(vertices(L), labels=tbc, cex=2)
# prune tree
tp <- treeprune(L, root=1, 1)
plot(tp, add=TRUE, col="blue", lwd=3)
```

triangulate.owin *Decompose Window into Triangles*

Description

Given a spatial window, this function decomposes the window into disjoint triangles. The result is a tessellation of the window in which each tile is a triangle.

Usage

```
triangulate.owin(W)
```

Arguments

W	Window (object of class "owin").

Details

The window W will be decomposed into disjoint triangles. The result is a tessellation of W in which each tile is a triangle. All triangle vertices lie on the boundary of the original polygon.

The window is first converted to a polygonal window using as.polygonal. The vertices of the polygonal window are extracted, and the Delaunay triangulation of these vertices is computed using delaunay. Each Delaunay triangle is intersected with the window: if the result is not a triangle, the triangulation procedure is applied recursively to this smaller polygon.

Value

Tessellation (object of class "tess").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

tess, delaunay, as.polygonal

Examples

```
plot(triangulate.owin(letterR))
```

trim.rectangle *Cut margins from rectangle*

Description

Trims a margin from a rectangle.

Usage

```
trim.rectangle(W, xmargin=0, ymargin=xmargin)
```

Arguments

W	A window (object of class "owin"). Must be of type "rectangle".
xmargin	Width of horizontal margin to be trimmed. A single nonnegative number, or a vector of length 2 indicating margins of unequal width at left and right.
ymargin	Height of vertical margin to be trimmed. A single nonnegative number, or a vector of length 2 indicating margins of unequal width at bottom and top.

Details

This is a simple convenience function to trim off a margin of specified width and height from each side of a rectangular window. Unequal margins can also be trimmed.

Value

Another object of class "owin" representing the window after margins are trimmed.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

grow.rectangle, erosion, owin.object

Examples

```
w <- square(10)
# trim a margin of width 1 from all four sides
square9 <- trim.rectangle(w, 1)

# trim margin of width 3 from the right side
# and margin of height 4 from top edge.
v <- trim.rectangle(w, c(0,3), c(0,4))
```

`triplet.family` *Triplet Interaction Family*

Description

An object describing the family of all Gibbs point processes with interaction order equal to 3.

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the interaction structure of Gibbs point processes which have infinite order of interaction, such as the triplet interaction process *Triplets*.

Anyway, `triplet.family` is an object of class `"isf"` containing a function `triplet.family$eval` for evaluating the sufficient statistics of a Gibbs point process model taking an exponential family form.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

See Also

`Triplets` to create the triplet interaction process structure.

Other families: `pairwise.family`, `pairsat.family`, `inforder.family`, `ord.family`.

`Triplets` *The Triplet Point Process Model*

Description

Creates an instance of Geyer's triplet interaction point process model which can then be fitted to point pattern data.

Usage

```
Triplets(r)
```

Arguments

`r`	The interaction radius of the Triplets process

Details

The (stationary) Geyer triplet process (Geyer, 1999) with interaction radius r and parameters β and γ is the point process in which each point contributes a factor β to the probability density of the point pattern, and each triplet of close points contributes a factor γ to the density. A triplet of close points is a group of 3 points, each pair of which is closer than r units apart.

Thus the probability density is

$$f(x_1, \ldots, x_n) = \alpha \beta^{n(x)} \gamma^{s(x)}$$

where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, $s(x)$ is the number of unordered triples of points that are closer than r units apart, and α is the normalising constant.

The interaction parameter γ must be less than or equal to 1 so that this model describes an "ordered" or "inhibitive" pattern.

The nonstationary Triplets process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location, rather than a constant beta.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Triplets process pairwise interaction is yielded by the function `Triplets()`. See the examples below.

Note the only argument is the interaction radius `r`. When `r` is fixed, the model becomes an exponential family. The canonical parameters $\log(\beta)$ and $\log(\gamma)$ are estimated by `ppm()`, not fixed in `Triplets()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the Triplets process with interaction radius r.

Author(s)

Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

`ppm`, `triplet.family`, `ppm.object`

Examples

```
Triplets(r=0.1)
# prints a sensible description of itself

## Not run:
ppm(cells, ~1, Triplets(r=0.2))
```

```
   # fit the stationary Triplets process to `cells'

## End(Not run)

   ppm(cells, ~polynom(x,y,3), Triplets(r=0.2))
   # fit a nonstationary Triplets process with log-cubic polynomial trend
```

Tstat *Third order summary statistic*

Description

Computes the third order summary statistic $T(r)$ of a spatial point pattern.

Usage

```
Tstat(X, ..., r = NULL, rmax = NULL,
    correction = c("border", "translate"), ratio = FALSE, verbose=TRUE)
```

Arguments

X	The observed point pattern, from which an estimate of $T(r)$ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp().
...	Ignored.
r	Optional. Vector of values for the argument r at which $T(r)$ should be evaluated. Users are advised *not* to specify this argument; there is a sensible default.
rmax	Optional. Numeric. The maximum value of r for which $T(r)$ should be estimated.
correction	Optional. A character vector containing any selection of the options "none", "border", "bord.modif", "translate", "translation", or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
verbose	Logical. If TRUE, an estimate of the computation time is printed.

Details

This command calculates the third-order summary statistic $T(r)$ for a spatial point patterns, defined by Schladitz and Baddeley (2000).

The definition of $T(r)$ is similar to the definition of Ripley's K function $K(r)$, except that $K(r)$ counts pairs of points while $T(r)$ counts triples of points. Essentially $T(r)$ is a rescaled cumulative distribution function of the diameters of triangles in the point pattern. The diameter of a triangle is the length of its longest side.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Computation time

If the number of points is large, the algorithm can take a very long time to inspect all possible triangles. A rough estimate of the total computation time will be printed at the beginning of the calculation. If this estimate seems very large, stop the calculation using the user interrupt signal, and call `Tstat` again, using `rmax` to restrict the range of `r` values, thus reducing the number of triangles to be inspected.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Schladitz, K. and Baddeley, A. (2000) A third order point process characteristic. *Scandinavian Journal of Statistics* **27** (2000) 657–671.

See Also

`Kest`

Examples

```
plot(Tstat(redwood))
```

`tweak.colourmap` *Change Colour Values in a Colour Map*

Description

Assign new colour values to some of the entries in a colour map.

Usage

```
tweak.colourmap(m, col, ..., inputs=NULL, range=NULL)
```

Arguments

`m`	A colour map (object of class `"colourmap"`).
`inputs`	Input values to the colour map, to be assigned new colours. Incompatible with `range`.
`range`	Numeric vector of length 2 specifying a range of numerical values which should be assigned a new colour. Incompatible with `inputs`.
`col`	Replacement colours for the specified `inputs` or the specified `range` of values.
`...`	Other arguments are ignored.

Details

This function changes the colour map `m` by assigning new colours to each of the input values specified by `inputs`, or by assigning a single new colour to the range of input values specified by `range`.

The modified colour map is returned.

Value

Another colour map (object of class "colourmap").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

colourmap, interp.colourmap, colouroutputs, colourtools.

Examples

```
co <- colourmap(rainbow(32), range=c(0,1))
plot(tweak.colourmap(co, inputs=c(0.5, 0.6), "white"))
plot(tweak.colourmap(co, range=c(0.5,0.6), "white"))
```

---

union.quad *Union of Data and Dummy Points*

Description

Combines the data and dummy points of a quadrature scheme into a single point pattern.

Usage

```
union.quad(Q)
```

Arguments

Q	A quadrature scheme (an object of class "quad").

Details

The argument Q should be a quadrature scheme (an object of class "quad", see quad.object for details).

This function combines the data and dummy points of Q into a single point pattern. If either the data or the dummy points are marked, the result is a marked point pattern.

The function as.ppp will perform the same task.

Value

A point pattern (of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

quad.object, as.ppp

Examples

```
data(simdat)
Q <- quadscheme(simdat, default.dummy(simdat))
U <- union.quad(Q)
## Not run: plot(U)
# equivalent:
U <- as.ppp(Q)
```

---

unique.ppp *Extract Unique Points from a Spatial Point Pattern*

Description

Removes any points that are identical to other points in a spatial point pattern.

Usage

```
## S3 method for class 'ppp'
unique(x, ..., warn=FALSE)

## S3 method for class 'ppx'
unique(x, ..., warn=FALSE)
```

Arguments

x	A spatial point pattern (object of class "ppp" or "ppx").
...	Arguments passed to duplicated.ppp or duplicated.data.frame.
warn	Logical. If TRUE, issue a warning message if any duplicated points were found.

Details

These are methods for the generic function unique for point pattern datasets (of class "ppp", see ppp.object, or class "ppx").

This function removes duplicate points in x, and returns a point pattern.

Two points in a point pattern are deemed to be identical if their x, y coordinates are the same, *and* their marks are the same (if they carry marks). This is the default rule: see duplicated.ppp for other options.

Value

Another point pattern object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppp.object, duplicated.ppp, multiplicity.ppp

Examples

```
X <- ppp(c(1,1,0.5), c(2,2,1), window=square(3))
unique(X)
unique(X, rule="deldir")
```

---

unitname *Name for Unit of Length*

Description

Inspect or change the name of the unit of length in a spatial dataset.

Usage

```
unitname(x)
## S3 method for class 'dppm'
unitname(x)
## S3 method for class 'im'
unitname(x)
## S3 method for class 'kppm'
unitname(x)
## S3 method for class 'minconfit'
unitname(x)
## S3 method for class 'owin'
unitname(x)
## S3 method for class 'ppm'
unitname(x)
## S3 method for class 'ppp'
unitname(x)
## S3 method for class 'psp'
unitname(x)
## S3 method for class 'quad'
unitname(x)
## S3 method for class 'slrm'
unitname(x)
## S3 method for class 'tess'
unitname(x)
unitname(x) <- value
## S3 replacement method for class 'dppm'
unitname(x) <- value
## S3 replacement method for class 'im'
unitname(x) <- value
## S3 replacement method for class 'kppm'
unitname(x) <- value
## S3 replacement method for class 'minconfit'
unitname(x) <- value
## S3 replacement method for class 'owin'
```

```
unitname(x) <- value
## S3 replacement method for class 'ppm'
unitname(x) <- value
## S3 replacement method for class 'ppp'
unitname(x) <- value
## S3 replacement method for class 'psp'
unitname(x) <- value
## S3 replacement method for class 'quad'
unitname(x) <- value
## S3 replacement method for class 'slrm'
unitname(x) <- value
## S3 replacement method for class 'tess'
unitname(x) <- value
```

Arguments

`x`	A spatial dataset. Either a point pattern (object of class "ppp"), a line segment pattern (object of class "psp"), a window (object of class "owin"), a pixel image (object of class "im"), a tessellation (object of class "tess"), a quadrature scheme (object of class "quad"), or a fitted point process model (object of class "ppm" or "kppm" or "slrm" or "dppm" or "minconfit").
`value`	Name of the unit of length. See Details.

Details

Spatial datasets in the **spatstat** package may include the name of the unit of length. This name is used when printing or plotting the dataset, and in some other applications.

`unitname(x)` extracts this name, and `unitname(x) <- value` sets the name to `value`.

A valid name is either

- a single character string
- a vector of two character strings giving the singular and plural forms of the unit name
- a list of length 3, containing two character strings giving the singular and plural forms of the basic unit, and a number specifying the multiple of this unit.

Note that re-setting the name of the unit of length *does not* affect the numerical values in `x`. It changes only the string containing the name of the unit of length. To rescale the numerical values, use `rescale`.

Value

The return value of `unitname` is an object of class "unitname" containing the name of the unit of length in `x`. There are methods for `print`, `summary`, `as.character`, `rescale` and `compatible`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`rescale`, `owin`, `ppp`

Examples

```
X <- runifpoint(20)

# if the unit of length is 1 metre:
unitname(X) <- c("metre", "metres")

# if the unit of length is 6 inches:
unitname(X) <- list("inch", "inches", 6)
```

---

unmark *Remove Marks*

Description

Remove the mark information from a spatial dataset.

Usage

```
unmark(X)
## S3 method for class 'ppp'
unmark(X)
## S3 method for class 'splitppp'
unmark(X)
## S3 method for class 'psp'
unmark(X)
## S3 method for class 'ppx'
unmark(X)
```

Arguments

X	A point pattern (object of class "ppp"), a split point pattern (object of class "splitppp"), a line segment pattern (object of class "psp") or a multidimensional space-time point pattern (object of class "ppx").

Details

A 'mark' is a value attached to each point in a spatial point pattern, or attached to each line segment in a line segment pattern, etc.

The function unmark is a simple way to remove the marks from such a dataset.

Value

An object of the same class as X with any mark information deleted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppp.object, psp.object

Examples

```
data(lansing)
hicks <- lansing[lansing$marks == "hickory", ]
## Not run:
plot(hicks)  # still a marked point pattern, but only 1 value of marks
plot(unmark(hicks)) # unmarked

## End(Not run)
```

`unnormdensity` *Weighted kernel smoother*

Description

An unnormalised version of kernel density estimation where the weights are not required to sum to 1. The weights may be positive, negative or zero.

Usage

```
unnormdensity(x, ..., weights = NULL)
```

Arguments

`x`	Numeric vector of data
`...`	Arguments passed to `density.default`. Arguments must be *named*. ‘
`weights`	Optional numeric vector of weights for the data.

Details

This is an alternative to the standard R kernel density estimation function `density.default`.

The standard `density.default` requires the `weights` to be nonnegative numbers that add up to 1, and returns a probability density (a function that integrates to 1).

This function `unnormdensity` does not impose any requirement on the `weights` except that they be finite. Individual weights may be positive, negative or zero. The result is a function that does not necessarily integrate to 1 and may be negative. The result is the convolution of the kernel k with the weighted data,

$$f(x) = \sum_i w_i k(x - x_i)$$

where x_i are the data points and w_i are the weights.

The algorithm first selects the kernel bandwidth by applying `density.default` to the data `x` with normalised, positive weight vector `w = abs(weights)/sum(abs(weights))` and extracting the selected bandwidth. Then the result is computed by applying applying `density.default` to `x` twice using the normalised positive and negative parts of the weights.

Note that the arguments `...` must be passed by name, i.e. in the form (`name=value`). Arguments that do not match an argument of `density.default` will be ignored *silently*.

Value

Object of class `"density"` as described in `density.default`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

density.default

Examples

```
d <- unnormdensity(1:3, weights=c(-1,0,1))
if(interactive()) plot(d)
```

---

unstack.msr *Separate a Vector Measure into its Scalar Components*

Description

Converts a vector-valued measure into a list of scalar-valued measures.

Usage

```
## S3 method for class 'msr'
unstack(x, ...)
```

Arguments

x	A measure (object of class "msr").
...	Ignored.

Details

This is a method for the generic unstack for the class "msr" of measures.

If x is a vector-valued measure, then y <- unstack(x) is a list of scalar-valued measures defined by the components of x. The jth entry of the list, y[[j]], is equivalent to the jth component of the vector measure x.

If x is a scalar-valued measure, then the result is a list consisting of one entry, which is x.

Value

A list of measures, of class "solist".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

unstack

unstack.ppp

split.msr.

Examples

```
fit <- ppm(cells ~ x)
m <- residuals(fit, type="score")
m
unstack(m)
```

---

unstack.ppp *Separate Multiple Columns of Marks*

Description

Given a spatial pattern with several columns of marks, take one column at a time, and return a list of spatial patterns each having only one column of marks.

Usage

```
## S3 method for class 'ppp'
unstack(x, ...)

## S3 method for class 'psp'
unstack(x, ...)

## S3 method for class 'lpp'
unstack(x, ...)
```

Arguments

x	A spatial point pattern (object of class "ppp" or "lpp") or a spatial pattern of line segments (object of class "psp").
...	Ignored.

Details

The functions defined here are methods for the generic unstack. The functions expect a spatial object x which has several columns of marks; they separate the columns, and return a list of spatial objects, each having only one column of marks.

If x has several columns of marks (i.e. marks(x) is a matrix, data frame or hyperframe with several columns), then y <- unstack(x) is a list of spatial objects, each of the same kind as x. The jth entry y[[j]] is equivalent to x except that it only includes the jth column of marks(x).

If x has no marks, or has only a single column of marks, the result is a list consisting of one entry, which is x.

Value

A list, of class "solist", whose entries are objects of the same type as x.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

unstack

unstack.msr

See also methods for the generic split such as split.ppp.

Examples

```
finpines
unstack(finpines)
```

---

update.detpointprocfamily
Set Parameter Values in a Determinantal Point Process Model

Description

Set parameter values in a determinantal point process model object.

Usage

```
## S3 method for class 'detpointprocfamily'
update(object, ...)
```

Arguments

object	object of class "detpointprocfamily".
...	arguments of the form tag=value specifying the parameters values to set.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

`update.interact` *Update an Interpoint Interaction*

Description

This command updates the `object` using the arguments given.

Usage

```
## S3 method for class 'interact'
update(object, ...)
```

Arguments

`object`	Interpoint interaction (object of class `"interact"`).
`...`	Additional or replacement values of parameters of `object`.

Details

This is a method for the generic function `update` for the class `"interact"` of interpoint interactions. It updates the `object` using the parameters given in the extra arguments `...`.

The extra arguments must be given in the form `name=value` and must be recognisable to the interaction object. They override any parameters of the same name in `object`.

Value

Another object of class `"interact"`, equivalent to `object` except for changes in parameter values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`update.ppm`

Examples

```
Str <- Strauss(r=1)
Str
update(Str, r=2)

M <- MultiStrauss(radii=matrix(1,2,2))
update(M, types=c("on", "off"))
```

update.kppm *Update a Fitted Cluster Point Process Model*

Description

update method for class "kppm".

Usage

```
## S3 method for class 'kppm'
update(object, ..., evaluate=TRUE)
```

Arguments

object	Fitted cluster point process model. An object of class "kppm", obtained from kppm.
...	Arguments passed to kppm.
evaluate	Logical value indicating whether to return the updated fitted model (evaluate=TRUE, the default) or just the updated call to kppm (evaluate=FALSE).

Details

object should be a fitted cluster point process model, obtained from the model-fitting function kppm. The model will be updated according to the new arguments provided.

If the argument trend is provided, it determines the intensity in the updated model. It should be an R formula (with or without a left hand side). It may include the symbols + or - to specify addition or deletion of terms in the current model formula, as shown in the Examples below. The symbol . refers to the current contents of the formula.

The intensity in the updated model is determined by the argument trend if it is provided, or otherwise by any unnamed argument that is a formula, or otherwise by the formula of the original model, formula(object).

The spatial point pattern data to which the new model is fitted is determined by the left hand side of the updated model formula, if this is present. Otherwise it is determined by the argument X if it is provided, or otherwise by any unnamed argument that is a point pattern or a quadrature scheme.

The model is refitted using kppm.

Value

Another fitted cluster point process model (object of class "kppm".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

kppm, plot.kppm, predict.kppm, simulate.kppm, methods.kppm, vcov.kppm

Examples

```
fit <- kppm(redwood ~1, "Thomas")
fitx <- update(fit, ~ . + x)
fitM <- update(fit, clusters="MatClust")
fitC <- update(fit, cells)
fitCx <- update(fit, cells ~ x)
```

---

update.ppm *Update a Fitted Point Process Model*

Description

update method for class "ppm".

Usage

```
## S3 method for class 'ppm'
update(object, ..., fixdummy=TRUE, use.internal=NULL,
                                    envir=environment(terms(object)))
```

Arguments

object	An existing fitted point process model, typically produced by ppm.
...	Arguments to be updated in the new call to ppm.
fixdummy	Logical flag indicating whether the quadrature scheme for the call to ppm should use the same set of dummy points as that in the original call.
use.internal	Optional. Logical flag indicating whether the model should be refitted using the internally saved data (use.internal=TRUE) or by re-evaluating these data in the current frame (use.internal=FALSE).
envir	Environment in which to re-evaluate the call to ppm.

Details

This is a method for the generic function update for the class "ppm". An object of class "ppm" describes a fitted point process model. See ppm.object) for details of this class.

update.ppm will modify the point process model specified by object according to the new arguments given, then re-fit it. The actual re-fitting is performed by the model-fitting function ppm.

If you are comparing several model fits to the same data, or fits of the same model to different data, it is strongly advisable to use update.ppm rather than trying to fit them by hand. This is because update.ppm re-fits the model in a way which is comparable to the original fit.

The arguments ... are matched to the formal arguments of ppm as follows.

First, all the *named* arguments in ... are matched with the formal arguments of ppm. Use name=NULL to remove the argument name from the call.

Second, any *unnamed* arguments in ... are matched with formal arguments of ppm if the matching is obvious from the class of the object. Thus ... may contain

- exactly one argument of class "ppp" or "quad", which will be interpreted as the named argument Q;

- exactly one argument of class "formula", which will be interpreted as the named argument trend (or as specifying a change to the trend formula);
- exactly one argument of class "interact", which will be interpreted as the named argument interaction;
- exactly one argument of class "data.frame", which will be interpreted as the named argument covariates.

The trend argument can be a formula that specifies a *change* to the current trend formula. For example, the formula ~ . + Z specifies that the additional covariate Z will be added to the right hand side of the trend formula in the existing object.

The argument fixdummy=TRUE ensures comparability of the objects before and after updating. When fixdummy=FALSE, calling update.ppm is exactly the same as calling ppm with the updated arguments. However, the original and updated models are not strictly comparable (for example, their pseudolikelihoods are not strictly comparable) unless they used the same set of dummy points for the quadrature scheme. Setting fixdummy=TRUE ensures that the re-fitting will be performed using the same set of dummy points. This is highly recommended.

The value of use.internal determines where to find data to re-evaluate the model (data for the arguments mentioned in the original call to ppm that are not overwritten by arguments to update.ppm).

If use.internal=FALSE, then arguments to ppm are *re-evaluated* in the frame where you call update.ppm. This is like the behaviour of the other methods for update. This means that if you have changed any of the objects referred to in the call, these changes will be taken into account. Also if the original call to ppm included any calls to random number generators, these calls will be recomputed, so that you will get a different outcome of the random numbers.

If use.internal=TRUE, then arguments to ppm are extracted from internal data stored inside the current fitted model object. This is useful if you don't want to re-evaluate anything. It is also necessary if if object has been restored from a dump file using load or source. In such cases, we have lost the environment in which object was fitted, and data cannot be re-evaluated.

By default, if use.internal is missing, update.ppm will re-evaluate the arguments if this is possible, and use internal data if not.

Value

Another fitted point process model (object of class "ppm").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
data(nztrees)
data(cells)

# fit the stationary Poisson process
fit <- ppm(nztrees, ~ 1)

# fit a nonstationary Poisson process
fitP <- update(fit, trend=~x)
fitP <- update(fit, ~x)

# change the trend formula: add another term to the trend
```

```
fitPxy <- update(fitP, ~ . + y)
# change the trend formula: remove the x variable
fitPy <- update(fitPxy, ~ . - x)

# fit a stationary Strauss process
fitS <- update(fit, interaction=Strauss(13))
fitS <- update(fit, Strauss(13))

# refit using a different edge correction
fitS <- update(fitS, correction="isotropic")

# re-fit the model to a subset
# of the original point pattern
nzw <- owin(c(0,148),c(0,95))
nzsub <- nztrees[,nzw]
fut <- update(fitS, Q=nzsub)
fut <- update(fitS, nzsub)

# WARNING: the point pattern argument is called 'Q'

ranfit <- ppm(rpoispp(42), ~1, Poisson())
ranfit
# different random data!
update(ranfit)
# the original data
update(ranfit, use.internal=TRUE)
```

---

update.rmhcontrol *Update Control Parameters of Metropolis-Hastings Algorithm*

Description

update method for class "rmhcontrol".

Usage

```
## S3 method for class 'rmhcontrol'
update(object, ...)
```

Arguments

object	Object of class "rmhcontrol" containing control parameters for a Metropolis-Hastings algorithm.
...	Arguments to be updated in the new call to rmhcontrol.

Details

This is a method for the generic function update for the class "rmhcontrol". An object of class "rmhcontrol" describes a set of control parameters for the Metropolis-Hastings simulation algorithm. See rmhcontrol).

update.rmhcontrol will modify the parameters specified by object according to the new arguments given.

Value

Another object of class "rmhcontrol".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
a <- rmhcontrol(expand=1)
update(a, expand=2)
```

---

update.symbolmap *Update a Graphics Symbol Map.*

Description

This command updates the object using the arguments given.

Usage

```
## S3 method for class 'symbolmap'
update(object, ...)
```

Arguments

object	Graphics symbol map (object of class "symbolmap").
...	Additional or replacement arguments to symbolmap.

Details

This is a method for the generic function update for the class "symbolmap" of graphics symbol maps. It updates the object using the parameters given in the extra arguments ....

The extra arguments must be given in the form name=value and must be recognisable to symbolmap. They override any parameters of the same name in object.

Value

Another object of class "symbolmap".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

See Also

symbolmap to create a graphics symbol map.

Examples

```
g <- symbolmap(size=function(x) x/50)
g
update(g, range=c(0,1))
update(g, size=42)
update(g, shape="squares", range=c(0,1))
```

`valid` *Check Whether Point Process Model is Valid*

Description

Determines whether a point process model object corresponds to a valid point process.

Usage

```
valid(object, ...)
```

Arguments

`object`	Object of some class, describing a point process model.
`...`	Additional arguments passed to methods.

Details

The function `valid` is generic, with methods for the classes `"ppm"` and `"dppmodel"`.

An object representing a point process is called valid if all its parameter values are known (for example, no parameter takes the value `NA` or `NaN`) and the parameter values correspond to a well-defined point process (for example, the parameter values satisfy all the constraints that are imposed by mathematical theory.)

See the methods for further details.

Value

A logical value, or `NA`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`valid.ppm`, `valid.detpointprocfamily`

valid.detpointprocfamily

Check Validity of a Determinantal Point Process Model

Description

Checks the validity of a determinantal point process model.

Usage

```
## S3 method for class 'detpointprocfamily'
valid(object, ...)
```

Arguments

object	Model of class "detpointprocfamily".
...	Ignored.

Value

Logical

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

valid

Examples

```
model1 <- dppMatern(lambda=100, alpha=.01, nu=1, d=2)
valid(model1)
model2 <- dppMatern(lambda=100, alpha=1, nu=1, d=2)
valid(model2)
```

valid.ppm *Check Whether Point Process Model is Valid*

Description

Determines whether a fitted point process model satisfies the integrability conditions for existence of the point process.

Usage

```
  ## S3 method for class 'ppm'
valid(object, warn=TRUE, ...)
```

Arguments

object	Fitted point process model (object of class "ppm").
warn	Logical value indicating whether to issue a warning if the validity of the model cannot be checked (due to unavailability of the required code).
...	Ignored.

Details

This is a method for the generic function valid for Poisson and Gibbs point process models (class "ppm").

The model-fitting function ppm fits Gibbs point process models to point pattern data. By default, ppm does not check whether the fitted model actually exists as a point process. This checking is done by valid.ppm.

Unlike a regression model, which is well-defined for any values of the fitted regression coefficients, a Gibbs point process model is only well-defined if the fitted interaction parameters satisfy some constraints. A famous example is the Strauss process (see Strauss) which exists only when the interaction parameter γ is less than or equal to 1. For values $\gamma > 1$, the probability density is not integrable and the process does not exist (and cannot be simulated).

By default, ppm does not enforce the constraint that a fitted Strauss process (for example) must satisfy $\gamma \leq 1$. This is because a fitted parameter value of $\gamma > 1$ could be useful information for data analysis, as it indicates that the Strauss model is not appropriate, and suggests a clustered model should be fitted.

The function valid.ppm checks whether the fitted model object specifies a well-defined point process. It returns TRUE if the model is well-defined.

Another possible reason for invalid models is that the data may not be adequate for estimation of the model parameters. In this case, some of the fitted coefficients could be NA or infinite values. If this happens then valid.ppm returns FALSE.

Use the function project.ppm to force the fitted model to be valid.

Value

A logical value, or NA.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, project.ppm

Examples

```
fit1 <- ppm(cells, ~1, Strauss(0.1))
valid(fit1)
fit2 <- ppm(redwood, ~1, Strauss(0.1))
valid(fit2)
```

---

varblock *Estimate Variance of Summary Statistic by Subdivision*

Description

This command estimates the variance of any summary statistic (such as the K-function) by spatial subdivision of a single point pattern dataset.

Usage

```
varblock(X, fun = Kest,
         blocks = quadrats(X, nx = nx, ny = ny),
         ...,
         nx = 3, ny = nx,
         confidence=0.95)
```

Arguments

X	Point pattern dataset (object of class "ppp").
fun	Function that computes the summary statistic.
blocks	Optional. A tessellation that specifies the division of the space into blocks.
...	Arguments passed to fun.
nx,ny	Optional. Number of rectangular blocks in the x and y directions. Incompatible with blocks.
confidence	Confidence level, as a fraction between 0 and 1.

Details

This command computes an estimate of the variance of the summary statistic fun(X) from a single point pattern dataset X using a subdivision method. It can be used to plot **confidence intervals** for the true value of a summary function such as the K-function.

The window containing X is divided into pieces by an nx * ny array of rectangles (or is divided into pieces of more general shape, according to the argument blocks if it is present). The summary statistic fun is applied to each of the corresponding sub-patterns of X as described below. Then the pointwise sample mean, sample variance and sample standard deviation of these summary statistics are computed. Then pointwise confidence intervals are computed, for the specified level of confidence, defaulting to 95 percent.

The variance is estimated by equation (4.21) of Diggle (2003, page 52). This assumes that the point pattern X is stationary. For further details see Diggle (2003, pp 52–53).

The estimate of the summary statistic from each block is computed as follows. For most functions fun, the estimate from block B is computed by finding the subset of X consisting of points that fall inside B, and applying fun to these points, by calling fun(X[B]).

However if fun is the K-function Kest, or any function which has an argument called domain, the estimate for each block B is computed by calling fun(X, domain=B). In the case of the K-function this means that the estimate from block B is computed by counting pairs of points in which the *first* point lies in B, while the second point may lie anywhere.

Value

A function value table (object of class "fv") that contains the result of fun(X) as well as the sample mean, sample variance and sample standard deviation of the block estimates, together with the upper and lower two-standard-deviation confidence limits.

Errors

If the blocks are too small, there may be insufficient data in some blocks, and the function fun may report an error. If this happens, you need to take larger blocks.

An error message about incompatibility may occur. The different function estimates may be incompatible in some cases, for example, because they use different default edge corrections (typically because the tiles of the tessellation are not the same kind of geometric object as the window of X, or because the default edge correction depends on the number of points). To prevent this, specify the choice of edge correction, in the correction argument to fun, if it has one.

An alternative to varblock is Loh's mark bootstrap lohboot.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

References

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

See Also

tess, quadrats for basic manipulation.

lohboot for an alternative bootstrap technique.

Examples

```
v <- varblock(amacrine, Kest, nx=4, ny=2)
v <- varblock(amacrine, Kcross, nx=4, ny=2)
if(interactive()) plot(v, iso ~ r, shade=c("hiiso", "loiso"))
```

varcount *Predicted Variance of the Number of Points*

Description

Given a fitted point process model, calculate the predicted variance of the number of points in a nominated set `B`.

Usage

```
varcount(model, B, ..., dimyx = NULL)
```

Arguments

`model`	A fitted point process model (object of class "ppm", "kppm" or "dppm").
`B`	A window (object of class "owin" specifying the region in which the points are counted. Alternatively a pixel image (object of class "im") or a function of spatial coordinates specifying a numerical weight for each random point.
`...`	Additional arguments passed to `B` when it is a function.
`dimyx`	Spatial resolution for the calculations. Argument passed to `as.mask`.

Details

This command calculates the variance of the number of points falling in a specified window `B` according to the `model`. It can also calculate the variance of a sum of weights attached to each random point.

The `model` should be a fitted point process model (object of class "ppm", "kppm" or "dppm").

- If `B` is a window, this command calculates the variance of the number of points falling in `B`, according to the fitted `model`.

  If the `model` depends on spatial covariates other than the Cartesian coordinates, then `B` should be a subset of the domain in which these covariates are defined.
- If `B` is a pixel image, this command calculates the variance of $T = \sum_i B(x_i)$, the sum of the values of `B` over all random points falling in the domain of the image.

  If the `model` depends on spatial covariates other than the Cartesian coordinates, then the domain of the pixel image, `as.owin(B)`, should be a subset of the domain in which these covariates are defined.
- If `B` is a `function(x,y)` or `function(x,y,...)` this command calculates the variance of $T = \sum_i B(x_i)$, the sum of the values of `B` over all random points falling inside the window `W=as.owin(model)`, the window in which the original data were observed.

The variance calculation involves the intensity and the pair correlation function of the model. The calculation is exact (up to discretisation error) for models of class "kppm" and "dppm", and for Poisson point process models of class "ppm". For Gibbs point process models of class "ppm" the calculation depends on the Poisson-saddlepoint approximations to the intensity and pair correlation function, which are rough approximations. The approximation is not yet implemented for some Gibbs models.

Value

A single number.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

predict.ppm, predict.kppm, predict.dppm

Examples

```
fitT <- kppm(redwood ~ 1, "Thomas")
B <- owin(c(0, 0.5), c(-0.5, 0))
varcount(fitT, B)

fitS <- ppm(swedishpines ~ 1, Strauss(9))
BS <- square(50)
varcount(fitS, BS)
```

---

vargamma.estK   *Fit the Neyman-Scott Cluster Point Process with Variance Gamma kernel*

Description

Fits the Neyman-Scott cluster point process, with Variance Gamma kernel, to a point pattern dataset by the Method of Minimum Contrast.

Usage

```
vargamma.estK(X, startpar=c(kappa=1,scale=1), nu = -1/4, lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...)
```

Arguments

X	Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the model.
nu	Numerical value controlling the shape of the tail of the clusters. A number greater than -1/2.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.

Details

This algorithm fits the Neyman-Scott Cluster point process model with Variance Gamma kernel (Jalilian et al, 2013) to a point pattern dataset by the Method of Minimum Contrast, using the K function.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The K function of the point pattern will be computed using Kest, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the K function, and this object should have been obtained by a call to Kest or one of its relatives.

The algorithm fits the Neyman-Scott Cluster point process with Variance Gamma kernel to X, by finding the parameters of the model which give the closest match between the theoretical K function of the model and the observed K function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The Neyman-Scott cluster point process with Variance Gamma kernel is described in Jalilian et al (2013). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent have a common distribution described in Jalilian et al (2013).

The shape of the kernel is determined by the dimensionless index nu. This is the parameter $\nu' = \alpha/2 - 1$ appearing in equation (12) on page 126 of Jalilian et al (2013). In previous versions of spatstat instead of specifying nu (called nu.ker at that time) the user could specify nu.pcf which is the parameter $\nu = \alpha - 1$ appearing in equation (13), page 127 of Jalilian et al (2013). These are related by nu.pcf = 2 * nu.ker + 1 and nu.ker = (nu.pcf - 1)/2. This syntax is still supported but not recommended for consistency across the package. In that case exactly one of nu.ker or nu.pcf must be specified.

If the argument lambda is provided, then this is used as the value of the point process intensity λ. Otherwise, if X is a point pattern, then λ will be estimated from X. If X is a summary statistic and lambda is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as NA.

The remaining arguments rmin,rmax,q,p control the method of minimum contrast; see mincontrast.

The corresponding model can be simulated using rVarGamma.

The parameter eta appearing in startpar is equivalent to the scale parameter omega used in rVarGamma.

Homogeneous or inhomogeneous Neyman-Scott/VarGamma models can also be fitted using the function kppm and the fitted models can be simulated using simulate.kppm.

The optimisation algorithm can be controlled through the additional arguments "..." which are passed to the optimisation function optim. For example, to constrain the parameter values to a certain range, use the argument method="L-BFGS-B" to select an optimisation algorithm that respects box constraints, and use the arguments lower and upper to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following main components:

`par`	Vector of fitted parameter values.
`fit`	Function value table (object of class `"fv"`) containing the observed values of the summary statistic (`observed`) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.ed

References

Jalilian, A., Guan, Y. and Waagepetersen, R. (2013) Decomposition of variance for spatial Cox processes. *Scandinavian Journal of Statistics* **40**, 119-137.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

`kppm`, `vargamma.estpcf`, `lgcp.estK`, `thomas.estK`, `cauchy.estK`, `mincontrast`, `Kest`, `Kmodel`.

`rVarGamma` to simulate the model.

Examples

```
if(interactive()) {
  u <- vargamma.estK(redwood)
  u
  plot(u)
}
```

`vargamma.estpcf` *Fit the Neyman-Scott Cluster Point Process with Variance Gamma kernel*

Description

Fits the Neyman-Scott cluster point process, with Variance Gamma kernel, to a point pattern dataset by the Method of Minimum Contrast, using the pair correlation function.

Usage

```
vargamma.estpcf(X, startpar=c(kappa=1,scale=1), nu = -1/4, lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL,
            ..., pcfargs = list())
```

Arguments

X	Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the model.
nu	Numerical value controlling the shape of the tail of the clusters. A number greater than -1/2.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.
pcfargs	Optional list containing arguments passed to pcf.ppp to control the smoothing in the estimation of the pair correlation function.

Details

This algorithm fits the Neyman-Scott Cluster point process model with Variance Gamma kernel (Jalilian et al, 2013) to a point pattern dataset by the Method of Minimum Contrast, using the pair correlation function.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The pair correlation function of the point pattern will be computed using pcf, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the pair correlation function, and this object should have been obtained by a call to pcf or one of its relatives.

The algorithm fits the Neyman-Scott Cluster point process with Variance Gamma kernel to X, by finding the parameters of the model which give the closest match between the theoretical pair correlation function of the model and the observed pair correlation function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The Neyman-Scott cluster point process with Variance Gamma kernel is described in Jalilian et al (2013). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent have a common distribution described in Jalilian et al (2013).

The shape of the kernel is determined by the dimensionless index nu. This is the parameter $\nu' = \alpha/2 - 1$ appearing in equation (12) on page 126 of Jalilian et al (2013). In previous versions of spatstat instead of specifying nu (called nu.ker at that time) the user could specify nu.pcf which is the parameter $\nu = \alpha - 1$ appearing in equation (13), page 127 of Jalilian et al (2013). These are related by `nu.pcf = 2 * nu.ker + 1` and `nu.ker = (nu.pcf - 1)/2`. This syntax is still supported but not recommended for consistency across the package. In that case exactly one of nu.ker or nu.pcf must be specified.

If the argument lambda is provided, then this is used as the value of the point process intensity λ. Otherwise, if X is a point pattern, then λ will be estimated from X. If X is a summary statistic and lambda is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as NA.

The remaining arguments rmin,rmax,q,p control the method of minimum contrast; see mincontrast.

The corresponding model can be simulated using rVarGamma.

The parameter eta appearing in startpar is equivalent to the scale parameter omega used in rVarGamma.

Homogeneous or inhomogeneous Neyman-Scott/VarGamma models can also be fitted using the function kppm and the fitted models can be simulated using simulate.kppm.

The optimisation algorithm can be controlled through the additional arguments "..." which are passed to the optimisation function optim. For example, to constrain the parameter values to a certain range, use the argument method="L-BFGS-B" to select an optimisation algorithm that respects box constraints, and use the arguments lower and upper to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following main components:

par Vector of fitted parameter values.

fit Function value table (object of class "fv") containing the observed values of the summary statistic (observed) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.ec

References

Jalilian, A., Guan, Y. and Waagepetersen, R. (2013) Decomposition of variance for spatial Cox processes. *Scandinavian Journal of Statistics* **40**, 119-137.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

kppm, vargamma.estK, lgcp.estpcf, thomas.estpcf, cauchy.estpcf, mincontrast, pcf, pcfmodel.

rVarGamma to simulate the model.

Examples

```
u <- vargamma.estpcf(redwood)
u
plot(u, legendpos="topright")
```

vcov.kppm *Variance-Covariance Matrix for a Fitted Cluster Point Process Model*

Description

Returns the variance-covariance matrix of the estimates of the parameters of a fitted cluster point process model.

Usage

```
## S3 method for class 'kppm'
vcov(object, ...,
        what=c("vcov", "corr", "fisher", "internals"),
        fast = NULL, rmax = NULL, eps.rmax = 0.01,
        verbose = TRUE)
```

Arguments

`object`	A fitted cluster point process model (an object of class `"kppm"`.)
`...`	Ignored.
`what`	Character string (partially-matched) that specifies what matrix is returned. Options are `"vcov"` for the variance-covariance matrix, `"corr"` for the correlation matrix, and `"fisher"` for the Fisher information matrix.
`fast`	Logical specifying whether tapering (using sparse matrices from **Matrix**) should be used to speed up calculations. Warning: This is expected to underestimate the true asymptotic variances/covariances.
`rmax`	Optional. The dependence range. Not usually specified by the user. Only used when `fast=TRUE`.
`eps.rmax`	Numeric. A small positive number which is used to determine `rmax` from the tail behaviour of the pair correlation function when fast option (`fast=TRUE`) is used. Namely `rmax` is the smallest value of r at which $(g(r)-1)/(g(0)-1)$ falls below `eps.rmax`. Only used when `fast=TRUE`. Ignored if `rmax` is provided.
`verbose`	Logical value indicating whether to print progress reports during very long calculations.

Details

This function computes the asymptotic variance-covariance matrix of the estimates of the canonical (regression) parameters in the cluster point process model `object`. It is a method for the generic function `vcov`.

The result is an `n * n` matrix where `n = length(coef(model))`.

To calculate a confidence interval for a regression parameter, use `confint` as shown in the examples.

Value

A square matrix.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Ported to **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Ege Rubak <rubak@math.aau.dk>.

References

Waagepetersen, R. (2007) Estimating functions for inhomogeneous spatial point processes with incomplete covariate data. *Biometrika* **95**, 351–363.

See Also

kppm, vcov, vcov.ppm

Examples

```
data(redwood)
fit <- kppm(redwood ~ x + y)
vcov(fit)
vcov(fit, what="corr")

# confidence interval
confint(fit)
# cross-check the confidence interval by hand:
sd <- sqrt(diag(vcov(fit)))
t(coef(fit) + 1.96 * outer(sd, c(lower=-1, upper=1)))
```

---

vcov.mppm — *Calculate Variance-Covariance Matrix for Fitted Multiple Point Process Model*

Description

Given a fitted multiple point process model, calculate the variance-covariance matrix of the parameter estimates.

Usage

```
    ## S3 method for class 'mppm'
vcov(object, ..., what="vcov", err="fatal")
```

Arguments

object	A multiple point process model (object of class "mppm").
...	Arguments recognised by vcov.ppm.
what	Character string indicating which quantity should be calculated. Options include "vcov" for the variance-covariance matrix, "corr" for the correlation matrix, and "fisher" for the Fisher information matrix.
err	Character string indicating what action to take if an error occurs. Either "fatal", "warn" or "null".

Details

This is a method for the generic function vcov.

The argument object should be a fitted multiple point process model (object of class "mppm") generated by mppm.

The variance-covariance matrix of the parameter estimates is computed using asymptotic theory for maximum likelihood (for Poisson processes) or estimating equations (for other Gibbs models).

If what="vcov" (the default), the variance-covariance matrix is returned. If what="corr", the variance-covariance matrix is normalised to yield a correlation matrix, and this is returned. If what="fisher", the Fisher information matrix is returned instead.

In all three cases, the rows and columns of the matrix correspond to the parameters (coefficients) in the same order as in coef{model}.

If errors or numerical problems occur, the argument err determines what will happen. If err="fatal" an error will occur. If err="warn" a warning will be issued and NA will be returned. If err="null", no warning is issued, but NULL is returned.

Value

A numeric matrix (or NA or NULL).

Error messages

An error message that reports *system is computationally singular* indicates that the determinant of the Fisher information matrix of one of the models was either too large or too small for reliable numerical calculation. See vcov.ppm for suggestions on how to handle this.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curti Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. London: Chapman and Hall/CRC Press.

See Also

vcov, vcov.ppm, mppm

Examples

```
fit <- mppm(Wat ~x, data=hyperframe(Wat=waterstriders))
vcov(fit)
```

vcov.ppm *Variance-Covariance Matrix for a Fitted Point Process Model*

Description

Returns the variance-covariance matrix of the estimates of the parameters of a fitted point process model.

Usage

```
  ## S3 method for class 'ppm'
vcov(object, ..., what = "vcov", verbose = TRUE,
                   fine=FALSE,
                   gam.action=c("warn", "fatal", "silent"),
                   matrix.action=c("warn", "fatal", "silent"),
                   logi.action=c("warn", "fatal", "silent"),
                   hessian=FALSE)
```

Arguments

`object`	A fitted point process model (an object of class "ppm".)
`...`	Ignored.
`what`	Character string (partially-matched) that specifies what matrix is returned. Options are "vcov" for the variance-covariance matrix, "corr" for the correlation matrix, and "fisher" or "Fisher" for the Fisher information matrix.
`fine`	Logical value indicating whether to use a quick estimate (`fine=FALSE`, the default) or a slower, more accurate estimate (`fine=TRUE`).
`verbose`	Logical. If `TRUE`, a message will be printed if various minor problems are encountered.
`gam.action`	String indicating what to do if `object` was fitted by `gam`.
`matrix.action`	String indicating what to do if the matrix is ill-conditioned (so that its inverse cannot be calculated).
`logi.action`	String indicating what to do if `object` was fitted via the logistic regression approximation using a non-standard dummy point process.
`hessian`	Logical. Use the negative Hessian matrix of the log pseudolikelihood instead of the Fisher information.

Details

This function computes the asymptotic variance-covariance matrix of the estimates of the canonical parameters in the point process model `object`. It is a method for the generic function `vcov`.

`object` should be an object of class "ppm", typically produced by `ppm`.

The canonical parameters of the fitted model `object` are the quantities returned by `coef.ppm(object)`. The function `vcov` calculates the variance-covariance matrix for these parameters.

The argument `what` provides three options:

`what="vcov"` return the variance-covariance matrix of the parameter estimates

`what="corr"` return the correlation matrix of the parameter estimates what="fisher" return the observed Fisher information matrix.

In all three cases, the result is a square matrix. The rows and columns of the matrix correspond to the canonical parameters given by coef.ppm(object). The row and column names of the matrix are also identical to the names in coef.ppm(object).

For models fitted by the Berman-Turner approximation (Berman and Turner, 1992; Baddeley and Turner, 2000) to the maximum pseudolikelihood (using the default method="mpl" in the call to ppm), the implementation works as follows.

- If the fitted model object is a Poisson process, the calculations are based on standard asymptotic theory for the maximum likelihood estimator (Kutoyants, 1998). The observed Fisher information matrix of the fitted model object is first computed, by summing over the Berman-Turner quadrature points in the fitted model. The asymptotic variance-covariance matrix is calculated as the inverse of the observed Fisher information. The correlation matrix is then obtained by normalising.
- If the fitted model is not a Poisson process (i.e. it is some other Gibbs point process) then the calculations are based on Coeurjolly and Rubak (2012). A consistent estimator of the variance-covariance matrix is computed by summing terms over all pairs of data points. If required, the Fisher information is calculated as the inverse of the variance-covariance matrix.

For models fitted by the Huang-Ogata method (method="ho" in the call to ppm), the implementation uses the Monte Carlo estimate of the Fisher information matrix that was computed when the original model was fitted.

For models fitted by the logistic regression approximation to the maximum pseudolikelihood (method="logi" in the call to ppm), calculations are based on (Baddeley et al., 2013). A consistent estimator of the variance-covariance matrix is computed by summing terms over all pairs of data points. If required, the Fisher information is calculated as the inverse of the variance-covariance matrix. In this case the calculations depend on the type of dummy pattern used, and currently only the types "stratrand", "binomial" and "poisson" as generated by quadscheme.logi are implemented. For other types the behavior depends on the argument logi.action. If logi.action="fatal" an error is produced. Otherwise, for types "grid" and "transgrid" the formulas for "stratrand" are used which in many cases should be conservative. For an arbitrary user specified dummy pattern (type "given") the formulas for "poisson" are used which in many cases should be conservative. If logi.action="warn" a warning is issued otherwise the calculation proceeds without a warning.

The argument verbose makes it possible to suppress some diagnostic messages.

The asymptotic theory is not correct if the model was fitted using gam (by calling ppm with use.gam=TRUE). The argument gam.action determines what to do in this case. If gam.action="fatal", an error is generated. If gam.action="warn", a warning is issued and the calculation proceeds using the incorrect theory for the parametric case, which is probably a reasonable approximation in many applications. If gam.action="silent", the calculation proceeds without a warning.

If hessian=TRUE then the negative Hessian (second derivative) matrix of the log pseudolikelihood, and its inverse, will be computed. For non-Poisson models, this is not a valid estimate of variance, but is useful for other calculations.

Note that standard errors and 95% confidence intervals for the coefficients can also be obtained using confint(object) or coef(summary(object)).

Value

A square matrix.

Error messages

An error message that reports *system is computationally singular* indicates that the determinant of the Fisher information matrix was either too large or too small for reliable numerical calculation.

If this message occurs, try repeating the calculation using `fine=TRUE`.

Singularity can occur because of numerical overflow or collinearity in the covariates. To check this, rescale the coordinates of the data points and refit the model. See the Examples.

In a Gibbs model, a singular matrix may also occur if the fitted model is a hard core process: this is a feature of the variance estimator.

Author(s)

Original code for Poisson point process was written by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz> . New code for stationary Gibbs point processes was generously contributed by Ege Rubak <rubak@math.aau.dk> and Jean-Francois Coeurjolly. New code for generic Gibbs process written by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>. New code for logistic method contributed by Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Coeurjolly, J.-F., Rubak, E. and Waagepetersen, R. (2014) Logistic regression for spatial Gibbs point processes. *Biometrika* **101** (2) 377–392.

Coeurjolly, J.-F. and Rubak, E. (2013) Fast covariance estimation for innovations computed from a spatial Gibbs point process. Scandinavian Journal of Statistics **40** 669–684.

Kutoyants, Y.A. (1998) **Statistical Inference for Spatial Poisson Processes**, Lecture Notes in Statistics 134. New York: Springer 1998.

See Also

`vcov` for the generic,

`ppm` for information about fitted models,

`confint` for confidence intervals.

Examples

```
  X <- rpoispp(42)
  fit <- ppm(X, ~ x + y)
  vcov(fit)
  vcov(fit, what="Fish")

  # example of singular system
  m <- ppm(demopat ~polynom(x,y,2))
  ## Not run:
    try(v <- vcov(m))

## End(Not run)
  # rescale x, y coordinates to range [0,1] x [0,1] approximately
  demopatScale <- rescale(demopat, 10000)
  m <- ppm(demopatScale ~ polynom(x,y,2))
  v <- vcov(m)

  # Gibbs example
  fitS <- ppm(swedishpines ~1, Strauss(9))
```

```
coef(fitS)
sqrt(diag(vcov(fitS)))
```

vcov.slrm *Variance-Covariance Matrix for a Fitted Spatial Logistic Regression*

Description

Returns the variance-covariance matrix of the estimates of the parameters of a point process model that was fitted by spatial logistic regression.

Usage

```
## S3 method for class 'slrm'
vcov(object, ...,
        what=c("vcov", "corr", "fisher", "Fisher"))
```

Arguments

`object`	A fitted point process model of class "slrm".
`...`	Ignored.
`what`	Character string (partially-matched) that specifies what matrix is returned. Options are "vcov" for the variance-covariance matrix, "corr" for the correlation matrix, and "fisher" or "Fisher" for the Fisher information matrix.

Details

This function computes the asymptotic variance-covariance matrix of the estimates of the canonical parameters in the point process model `object`. It is a method for the generic function `vcov`.

`object` should be an object of class "slrm", typically produced by `slrm`. It represents a Poisson point process model fitted by spatial logistic regression.

The canonical parameters of the fitted model `object` are the quantities returned by `coef.slrm(object)`. The function `vcov` calculates the variance-covariance matrix for these parameters.

The argument `what` provides three options:

`what="vcov"` return the variance-covariance matrix of the parameter estimates

`what="corr"` return the correlation matrix of the parameter estimates

`what="fisher"` return the observed Fisher information matrix.

In all three cases, the result is a square matrix. The rows and columns of the matrix correspond to the canonical parameters given by `coef.slrm(object)`. The row and column names of the matrix are also identical to the names in `coef.slrm(object)`.

Note that standard errors and 95% confidence intervals for the coefficients can also be obtained using `confint(object)` or `coef(summary(object))`.

Standard errors for the fitted intensity can be obtained using `predict.slrm`.

Value

A square matrix.

Error messages

An error message that reports *system is computationally singular* indicates that the determinant of the Fisher information matrix was either too large or too small for reliable numerical calculation. This can occur because of numerical overflow or collinearity in the covariates.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz> .

References

Baddeley, A., Berman, M., Fisher, N.I., Hardegen, A., Milne, R.K., Schuhmacher, D., Shah, R. and Turner, R. (2010) Spatial logistic regression and change-of-support for spatial Poisson point processes. *Electronic Journal of Statistics* **4**, 1151–1201. doi: 10.1214/10-EJS581

See Also

vcov for the generic,

slrm for information about fitted models,

predict.slrm for other kinds of calculation about the model,

confint for confidence intervals.

Examples

```
X <- rpoispp(42)
fit <- slrm(X ~ x + y)
vcov(fit)
vcov(fit, what="corr")
vcov(fit, what="f")
```

---

vertices — *Vertices of a Window*

Description

Finds the vertices of a window, or similar object.

Usage

```
 vertices(w)

 ## S3 method for class 'owin'
vertices(w)
```

Arguments

w	A window (object of class "owin") or similar object.

Details

This function computes the vertices ('corners') of a spatial window or other object.

For `vertices.owin`, the argument `w` should be a window (an object of class `"owin"`, see `owin.object` for details).

If `w` is a rectangle, the coordinates of the four corner points are returned.

If `w` is a polygonal window (consisting of one or more polygons), the coordinates of the vertices of all polygons are returned.

If `w` is a binary mask, then a 'boundary pixel' is defined to be a pixel inside the window which has at least one neighbour outside the window. The coordinates of the centres of all boundary pixels are returned.

Value

A list with components `x` and `y` giving the coordinates of the vertices.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`owin.object`.

Examples

```
data(letterR)
vert <- vertices(letterR)

plot(letterR, main="Polygonal vertices")
points(vert)
plot(letterR, main="Boundary pixels")
points(vertices(as.mask(letterR)))
```

---

volume — *Volume of an Object*

Description

Computes the volume of a spatial object such as a three-dimensional box.

Usage

```
volume(x)
```

Arguments

`x`	An object whose volume will be computed.

Details

This function computes the volume of an object such as a three-dimensional box.

The function `volume` is generic, with methods for the classes `"box3"` (three-dimensional boxes) and `"boxx"` (multi-dimensional boxes).

There is also a method for the class `"owin"` (two-dimensional windows), which is identical to `area.owin`, and a method for the class `"linnet"` of linear networks, which returns the length of the network.

Value

The numerical value of the volume of the object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`area.owin`, `volume.box3`, `volume.boxx`, `volume.linnet`

`weighted.median` *Weighted Median, Quantiles or Variance*

Description

Compute the median, quantiles or variance of a set of numbers which have weights associated with them.

Usage

```
weighted.median(x, w, na.rm = TRUE)

weighted.quantile(x, w, probs=seq(0,1,0.25), na.rm = TRUE)

weighted.var(x, w, na.rm = TRUE)
```

Arguments

`x`	Data values. A vector of numeric values, for which the median or quantiles are required.
`w`	Weights. A vector of nonnegative numbers, of the same length as `x`.
`probs`	Probabilities for which the quantiles should be computed. A numeric vector of values between 0 and 1.
`na.rm`	Logical. Whether to ignore `NA` values.

Details

The ith observation x[i] is treated as having a weight proportional to w[i].

The weighted median is a value m such that the total weight of data to the left of m is equal to half the total weight. If there is no such value, linear interpolation is performed.

Value

A numeric value or vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

quantile, median.

Examples

```
x <- 1:20
w <- runif(20)
weighted.median(x, w)
weighted.quantile(x, w)
weighted.var(x, w)
```

---

where.max *Find Location of Maximum in a Pixel Image*

Description

Finds the spatial location(s) where a given pixel image attains its maximum or minimum value.

Usage

```
where.max(x, first = TRUE)
where.min(x, first = TRUE)
```

Arguments

x	A pixel image (object of class "im").
first	Logical value. If TRUE (the default), then only one location will be returned. If FALSE, then all locations where the maximum is achieved will be returned.

Details

This function finds the spatial location or locations where the pixel image x attains its maximum or minimum value. The result is a point pattern giving the locations.

If first=TRUE (the default), then only one location will be returned, namely the location with the smallest y coordinate value which attains the maximum or minimum. This behaviour is analogous to the functions which.min and which.max.

If first=FALSE, then the function returns the locations of all pixels where the maximum (or minimum) value is attained. This could be a large number of points.

Value

A point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

Summary.im for computing the minimum and maximum of pixel values; eval.im and Math.im for mathematical expressions involving images; solutionset for finding the set of pixels where a statement is true.

Examples

```
D <- distmap(letterR, invert=TRUE)
plot(D)
plot(where.max(D), add=TRUE, pch=16, cols="green")
```

---

whichhalfplane *Test Which Side of Infinite Line a Point Falls On*

Description

Given an infinite line and a spatial point location, determine which side of the line the point falls on.

Usage

```
whichhalfplane(L, x, y = NULL)
```

Arguments

L	Object of class "infline" specifying one or more infinite straight lines in two dimensions.
x,y	Arguments acceptable to xy.coords specifying the locations of the points.

Details

An infinite line L divides the two-dimensional plane into two half-planes. This function returns a matrix M of logical values in which M[i,j] = TRUE if the jth spatial point lies below or to the left of the ith line.

Value

A logical matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

infline

Examples

```
L <- infline(p=runif(3), theta=runif(3, max=2*pi))
X <- runifpoint(4)
whichhalfplane(L, X)
```

---

whist *Weighted Histogram*

Description

Computes the weighted histogram of a set of observations with a given set of weights.

Usage

```
whist(x, breaks, weights = NULL)
```

Arguments

x	Numeric vector of observed values.
breaks	Vector of breakpoints for the histogram.
weights	Numeric vector of weights for the observed values.

Details

This low-level function computes (but does not plot) the weighted histogram of a vector of observations x using a given vector of weights.

The arguments x and weights should be numeric vectors of equal length. They may include NA or infinite values.

The argument breaks should be a numeric vector whose entries are strictly increasing. These values define the boundaries between the successive histogram cells. The breaks *do not* have to span the range of the observations.

There are N-1 histogram cells, where N = length(breaks). An observation x[i] falls in the jth cell if breaks[j] <= x[i] < breaks[j+1] (for j < N-1) or breaks[j] <= x[i] <= breaks[j+1] (for j = N-1). The weighted histogram value h[j] for the jth cell is the sum of weights[i] for all observations x[i] that fall in the cell.

Note that, in contrast to the function hist, the function whist does not require the breakpoints to span the range of the observations x. Values of x that fall outside the range of breaks are handled separately; their total weight is returned as an attribute of the histogram.

Value

A numeric vector of length N-1 containing the histogram values, where N = length(breaks).

The return value also has attributes "low" and "high" giving the total weight of all observations that are less than the lowest breakpoint, or greater than the highest breakpoint, respectively.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

with thanks to Peter Dalgaard.

Examples

```
x <- rnorm(100)
b <- seq(-1,1,length=21)
w <- runif(100)
whist(x,b,w)
```

---

will.expand *Test Expansion Rule*

Description

Determines whether an expansion rule will actually expand the window or not.

Usage

```
will.expand(x)
```

Arguments

x	Expansion rule. An object of class "rmhexpand".

Details

An object of class "rmhexpand" describes a rule for expanding a simulation window. See rmhexpand for details.

One possible expansion rule is to do nothing, i.e. not to expand the window.

This command inspects the expansion rule x and determines whether it will or will not actually expand the window. It returns TRUE if the window will be expanded.

Value

Logical value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

rmhexpand, expand.owin

Examples

```
x <- rmhexpand(distance=0.2)
y <- rmhexpand(area=1)
will.expand(x)
will.expand(y)
```

Window *Extract or Change the Window of a Spatial Object*

Description

Given a spatial object (such as a point pattern or pixel image) in two dimensions, these functions extract or change the window in which the object is defined.

Usage

```
Window(X, ...)

Window(X, ...) <- value

## S3 method for class 'ppp'
Window(X, ...)

## S3 replacement method for class 'ppp'
Window(X, ...) <- value

## S3 method for class 'psp'
Window(X, ...)

## S3 replacement method for class 'psp'
Window(X, ...) <- value

## S3 method for class 'im'
Window(X, ...)

## S3 replacement method for class 'im'
Window(X, ...) <- value
```

Arguments

X	A spatial object such as a point pattern, line segment pattern or pixel image.
...	Extra arguments. They are ignored by all the methods listed here.
value	Another window (object of class "owin") to be used as the window for X.

Details

The functions Window and Window<- are generic.

Window(X) extracts the spatial window in which X is defined.

Window(X) <- W changes the window in which X is defined to the new window W, and *discards any data outside* W. In particular:

- If X is a point pattern (object of class "ppp") then Window(X) <- W discards any points of X which fall outside W.
- If X is a line segment pattern (object of class "psp") then Window(X) <- W clips the segments of X to the boundaries of W.
- If X is a pixel image (object of class "im") then Window(X) <- W has the effect that pixels lying outside W are retained but their pixel values are set to NA.

Many other classes of spatial object have a method for Window, but not Window<-. See Window.ppm.

Value

The result of Window is a window (object of class "owin").

The result of Window<- is the updated object X, of the same class as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

Window.ppm

Examples

```
## point patterns
Window(cells)
X <- demopat
Window(X)
Window(X) <- as.rectangle(Window(X))

## line segment patterns
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
Window(X)
Window(X) <- square(0.5)

## images
Z <- setcov(owin())
Window(Z)
Window(Z) <- square(0.5)
```

WindowOnly *Extract Window of Spatial Object*

Description

Given a spatial object (such as a point pattern or pixel image) in two dimensions, these functions extract the window in which the object is defined.

Usage

```
## S3 method for class 'ppm'
Window(X, ..., from=c("points", "covariates"))

## S3 method for class 'kppm'
Window(X, ..., from=c("points", "covariates"))

## S3 method for class 'dppm'
Window(X, ..., from=c("points", "covariates"))

## S3 method for class 'lpp'
Window(X, ...)

## S3 method for class 'lppm'
Window(X, ...)

## S3 method for class 'msr'
Window(X, ...)

## S3 method for class 'quad'
Window(X, ...)

## S3 method for class 'quadratcount'
Window(X, ...)

## S3 method for class 'quadrattest'
Window(X, ...)

## S3 method for class 'tess'
Window(X, ...)

## S3 method for class 'layered'
Window(X, ...)

## S3 method for class 'distfun'
Window(X, ...)

## S3 method for class 'nnfun'
Window(X, ...)

## S3 method for class 'funxy'
Window(X, ...)

## S3 method for class 'rmhmodel'
Window(X, ...)

## S3 method for class 'leverage.ppm'
Window(X, ...)

## S3 method for class 'influence.ppm'
Window(X, ...)
```

Arguments

X	A spatial object.
...	Ignored.
from	Character string. See Details.

Details

These are methods for the generic function Window which extract the spatial window in which the object X is defined.

The argument from applies when X is a fitted point process model (object of class "ppm", "kppm" or "dppm"). If from="data" (the default), Window extracts the window of the original point pattern data to which the model was fitted. If from="covariates" then Window returns the window in which the spatial covariates of the model were provided.

Value

An object of class "owin" (see owin.object) specifying an observation window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

Window, Window.ppp, Window.psp.

owin.object

Examples

```
X <- quadratcount(cells, 4)
Window(X)
```

---

with.fv *Evaluate an Expression in a Function Table*

Description

Evaluate an R expression in a function value table (object of class "fv").

Usage

```
## S3 method for class 'fv'
with(data, expr, ..., fun = NULL, enclos=NULL)
```

Arguments

data	A function value table (object of class "fv") in which the expression will be evaluated.
expr	The expression to be evaluated. An R language expression, which may involve the names of columns in data, the special abbreviations ., .x and .y, and global constants or functions.
...	Ignored.
fun	Logical value, specifying whether the result should be interpreted as another function (fun=TRUE) or simply returned as a numeric vector or array (fun=FALSE). See Details.
enclos	An environment in which to search for variables that are not found in data. Defaults to parent.frame().

Details

This is a method for the generic command with for an object of class "fv" (function value table).

An object of class "fv" is a convenient way of storing and plotting several different estimates of the same function. It is effectively a data frame with extra attributes. See fv.object for further explanation.

This command makes it possible to perform computations that involve different estimates of the same function. For example we use it to compute the arithmetic difference between two different edge-corrected estimates of the K function of a point pattern.

The argument expr should be an R language expression. The expression may involve

- the name of any column in data, referring to one of the estimates of the function;
- the symbol . which stands for all the available estimates of the function;
- the symbol .y which stands for the recommended estimate of the function (in an "fv" object, one of the estimates is always identified as the recommended estimate);
- the symbol .x which stands for the argument of the function;
- global constants or functions.

See the Examples. The expression should be capable of handling vectors and matrices.

The interpretation of the argument fun is as follows:

- If fun=FALSE, the result of evaluating the expression expr will be returned as a numeric vector, matrix or data frame.
- If fun=TRUE, then the result of evaluating expr will be interpreted as containing the values of a new function. The return value will be an object of class "fv". (This can only happen if the result has the right dimensions.)
- The default is fun=TRUE if the result of evaluating expr has more than one column, and fun=FALSE otherwise.

To perform calculations involving *several* objects of class "fv", use eval.fv.

Value

A function value table (object of class "fv") or a numeric vector or data frame.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

with, fv.object, eval.fv, Kest

Examples

```
# compute 4 estimates of the K function
X <- rpoispp(42)
K <- Kest(X)
plot(K)

# derive 4 estimates of the L function L(r) = sqrt(K(r)/pi)
L <- with(K, sqrt(./pi))
plot(L)

# compute 4 estimates of V(r) = L(r)/r
V <- with(L, ./.x)
plot(V)

# compute the maximum absolute difference between
# the isotropic and translation correction estimates of K(r)
D <- with(K, max(abs(iso - trans)))
```

`with.hyperframe` *Evaluate an Expression in Each Row of a Hyperframe*

Description

An expression, involving the names of columns in a hyperframe, is evaluated separately for each row of the hyperframe.

Usage

```
## S3 method for class 'hyperframe'
with(data, expr, ...,
                          simplify = TRUE,
                          ee = NULL, enclos=NULL)
```

Arguments

data	A hyperframe (object of class "hyperframe") containing data.
expr	An R language expression to be evaluated.
...	Ignored.
simplify	Logical. If TRUE, the return value will be simplified to a vector whenever possible.
ee	Alternative form of expr, as an object of class "expression".
enclos	An environment in which to search for objects that are not found in the hyperframe. Defaults to parent.frame().

Details

This function evaluates the expression `expr` in each row of the hyperframe `data`. It is a method for the generic function `with`.

The argument `expr` should be an R language expression in which each variable name is either the name of a column in the hyperframe `data`, or the name of an object in the parent frame (the environment in which `with` was called.) The argument `ee` can be used as an alternative to `expr` and should be an expression object (of class `"expression"`).

For each row of `data`, the expression will be evaluated so that variables which are column names of `data` are interpreted as the entries for those columns in the current row.

For example, if a hyperframe `h` has columns called `A` and `B`, then `with(h, A != B)` inspects each row of `data` in turn, tests whether the entries in columns `A` and `B` are equal, and returns the n logical values.

Value

Normally a list of length n (where n is the number of rows) containing the results of evaluating the expression for each row. If `simplify=TRUE` and each result is a single atomic value, then the result is a vector or factor containing the same values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`hyperframe`, `plot.hyperframe`

Examples

```
# generate Poisson point patterns with intensities 10 to 100
 H <- hyperframe(L=seq(10,100, by=10))
 X <- with(H, rpoispp(L))
```

`with.msr` *Evaluate Expression Involving Components of a Measure*

Description

An expression involving the names of components of a measure is evaluated.

Usage

```
## S3 method for class 'msr'
with(data, expr, ...)
```

Arguments

`data`	A measure (object of class `"msr"`).
`expr`	An expression to be evaluated.
`...`	Ignored.

Details

This is a method for the generic function with for the class "msr". The argument data should be an object of class "msr" representing a measure (a function which assigns a value to each subset of two-dimensional space).

This function can be used to extract the components of the measure, or to perform more complicated manipulations of the components.

The argument expr should be an un-evaluated expression in the R language. The expression may involve any of the variable names listed below with their corresponding meanings.

qlocations	(point pattern) all quadrature locations
qweights	(numeric) all quadrature weights
density	(numeric) density value at each quadrature point
discrete	(numeric) discrete mass at each quadrature point
continuous	(numeric) increment of continuous component
increment	(numeric) increment of measure
is.atom	(logical) whether quadrature point is an atom
atoms	(point pattern) locations of atoms
atommass	(numeric) massess of atoms

The measure is the sum of discrete and continuous components. The discrete component assigns non-zero mass to several points called atoms. The continuous component has a density which should be integrated over a region to determine the value for that region.

An object of class "msr" approximates the continuous component by a sum over quadrature points. The quadrature points are chosen so that they include the atoms of the measure. In the list above, we have increment = continuous + discrete, continuous = density * qweights, is.atom = (discrete > 0), atoms = qlocations[is.atom] and atommass = discrete[is.atom].

Value

The result of evaluating the expression could be an object of any kind.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

msr, split.msr

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
rp <- residuals(fit, type="pearson")

with(rp, atoms)
with(rp, qlocations %mark% continuous)
```

with.ssf *Evaluate Expression in a Spatially Sampled Function*

Description

Given a spatially sampled function, evaluate an expression involving the function values.

Usage

```
apply.ssf(X, ...)

## S3 method for class 'ssf'
with(data, ...)
```

Arguments

X, data	A spatially sampled function (object of class "ssf").
...	Arguments passed to with.default or apply specifying what to compute.

Details

An object of class "ssf" represents a function (real- or vector-valued) that has been sampled at a finite set of points. It contains a data frame which provides the function values at the sample points.

In with.ssf, the expression specified by ... will be evaluated in this dataframe. In apply.ssf, the dataframe will be subjected to the apply operator using the additional arguments ....

If the result of evaluation is a data frame with one row for each data point, or a numeric vector with one entry for each data point, then the result will be an object of class "ssf" containing this information. Otherwise, the result will be a numeric vector.

Value

An object of class "ssf" or a numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

ssf

Examples

```
a <- ssf(cells, data.frame(d=nndist(cells), i=1:npoints(cells)))
with(a, i/d)
```

yardstick	*Text, Arrow or Scale Bar in a Diagram*

Description

Create spatial objects that represent a text string, an arrow, or a yardstick (scale bar).

Usage

```
textstring(x, y, txt = NULL, ...)

onearrow(x0, y0, x1, y1, txt = NULL, ...)

yardstick(x0, y0, x1, y1, txt = NULL, ...)
```

Arguments

`x,y`	Coordinates where the text should be placed.
`x0,y0,x1,y1`	Spatial coordinates of both ends of the arrow or yardstick. Alternatively `x0` can be a point pattern (class `"ppp"`) containing exactly two points, or a line segment pattern (class `"psp"`) consisting of exactly one line segment.
`txt`	The text to be displayed beside the line segment. Either a character string or an expression.
`...`	Additional named arguments for plotting the object.

Details

These commands create objects that represent components of a diagram:

- `textstring` creates an object that represents a string of text at a particular spatial location.
- `onearrow` creates an object that represents an arrow between two locations.
- `yardstick` creates an object that represents a scale bar: a line segment indicating the scale of the plot.

To display the relevant object, it should be plotted, using `plot`. See the help files for the plot methods `plot.textstring`, `plot.onearrow` and `plot.yardstick`.

These objects are designed to be included as components in a `layered` object or a `solist`. This makes it possible to build up a diagram consisting of many spatial objects, and to annotate the diagram with arrows, text and so on, so that ultimately the entire diagram is plotted using `plot`.

Value

An object of class `"diagramobj"` which also belongs to one of the special classes `"textstring"`, `"onearrow"` or `"yardstick"`. There are methods for `plot`, `print`, `"["` and `shift`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

plot.textstring, plot.onearrow, plot.yardstick.

Examples

```
X <- rescale(swedishpines)
plot(X, pch=16, main="")
ys <- yardstick(as.psp(list(xmid=4, ymid=0.5, length=1, angle=0),
                       window=Window(X)),
                txt="1 m")
plot(ys, angle=90)
```

---

zapsmall.im *Rounding of Pixel Values*

Description

Modifies a pixel image, identifying those pixels that have values very close to zero, and replacing the value by zero.

Usage

```
zapsmall.im(x, digits)
```

Arguments

x	Pixel image (object of class "im").
digits	Argument passed to zapsmall indicating the precision to be used.

Details

The function zapsmall is applied to each pixel value of the image x.

Value

Another pixel image.

Author(s)

Ege Rubak <rubak@math.aau.dk> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

zapsmall

Examples

```
data(cells)
D <- density(cells)
zapsmall.im(D)
```

zclustermodel *Cluster Point Process Model*

Description

Experimental code. Creates an object representing a cluster point process model. Typically used for theoretical calculations about such a model.

Usage

```
zclustermodel(name = "Thomas", ..., mu, kappa, scale)
```

Arguments

name	Name of the cluster process. One of "Thomas", "MatClust", "VarGamma" or "Cauchy".
...	Other arguments needed for the model.
mu	Mean cluster size. A single number, or a pixel image.
kappa	Parent intensity. A single number.
scale	Cluster scale parameter of the model.

Details

Experimental.

Value

Object of the experimental class "zclustermodel".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

methods.zclustermodel

Examples

```
m <- zclustermodel("Thomas", kappa=10, mu=5, scale=0.1)
```

[.ssf *Subset of spatially sampled function*

Description

Extract a subset of the data for a spatially sampled function.

Usage

```
## S3 method for class 'ssf'
x[i, j, ..., drop]
```

Arguments

x	Object of class "ssf".
i	Subset index applying to the locations where the function is sampled.
j	Subset index applying to the columns (variables) measured at each location.
..., drop	Ignored.

Details

This is the subset operator for the class "ssf".

Value

Another object of class "ssf".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

ssf, with.ssf

Examples

```
f <- ssf(cells, data.frame(d=nndist(cells), i=1:42))
f
f[1:10,]
f[ ,1]
```

Index