Package ‘mvp’

January 30, 2019

Type Package

Title Fast Symbolic Multivariate Polynomials

Version 1.0-2

Date 2019-01-11

Author Robin K. S. Hankin

Depends methods

Suggests knitr,rmarkdown,spray

VignetteBuilder knitr

Maintainer Robin K. S. Hankin <hankin.robin@gmail.com>

Description Fast manipulation of symbolic multivariate polynomials
using the 'Map' class of the Standard Template Library. The package
uses print and coercion methods from the 'mpoly' package (Kahle 2013,
``Multivariate polynomials in R''. The R Journal, 5(1):162), but offers
speed improvements. It is comparable in speed to the 'spray' package
for sparse arrays, but retains the symbolic benefits of 'mpoly'.

License GPL (>= 2)

Imports Rcpp (>= 0.12.3),partitions,mpoly,magic

LinkingTo Rcpp

SystemRequirements C++11

URL https://github.com/RobinHankin/mvp.git

BugReports https://github.com/RobinHankin/mvp/issues

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-01-14 12:50:03 UTC

R topics documented:

accessor *Accessor methods for mvp objects*

Description

Accessor methods for mvp objects

Usage

```
vars(x)
powers(x)
coeffs(x)
coeffs(x) <- value
```

Arguments

x	Object of class mvp
value	Numeric vector of length 1

Details

Access the different parts of an mvp object. The constant term is technically a coefficient but is documented under `constant.Rd`.

Note

The terms of an `mvp` object are not held in any particular order; the order of the terms is not defined. This is because the `map` class of the `STL` does not specify an order for the key-value pairs (which may be implementation dependent). The situation is similar to the `hyper2` package which uses a similar scheme.

So the output of `coeffs(x)` is defined only up to an unknown rearrangement. If all the coefficients are the same, this does not matter. The same considerations apply to the output of `vars()` and the output of `powers()` (which return lists whose elements are in an undefined order).

However, even though the order of these three objects is undefined individually, their ordering is jointly consistent in the sense that the first element of `coeffs(x)` corresponds to the first element of `vars(x)` and the first element of `powers(x)`. The identity of this element is not defined—but whatever it is, the first element of all three accessor methods refers to it.

The vignette discusses this.

Author(s)

Robin K. S. Hankin

See Also

`constant`

Examples

```
a <- rmvp(5)
vars(a)
powers(a)
coeffs(a)

coeffs(a) <- 1  # A simpler object
coeffs(a) <- 0  # The zero polynomial
```

`allvars` *All variables in a multivariate polynomial*

Description

Returns a character vector containing all the variables present in a mvp object

Usage

```
allvars(x)
```

Arguments

`x`	object of class mvp

Note

The character vector returned is not in any particular order

Author(s)

Robin K. S. Hankin

Examples

```
p <- rmvp(5)
allvars(p)
```

as.function.mvp *Functional form for multivariate polynomials*

Description

Coerces a multivariate polynomial into a function

Usage

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

Arguments

`x`	Multivariate polynomial
`...`	Further arguments (currently ignored)

Author(s)

Robin K. S. Hankin

Examples

```

p <- as.mvp("1+a^2 + a*b^2 + c")
p
f <- as.function(p)

f(a=1)
f(a=1,b=2)
f(a=1,b=2,c=3)
f(a=1,b=2,c=3,drop=FALSE)
```

constant *The constant term*

Description

Get and set the constant term of a mvp object

Usage

```
## S3 method for class 'mvp'
constant(x)
## S3 replacement method for class 'mvp'
constant(x) <- value
## S3 method for class 'numeric'
constant(x)
```

Arguments

`x`	Object of class mvp
`value`	Scalar value for the constant

Details

The constant term in a polynomial is the coefficient of the empty term. In an mvp object, the map `{} -> c`, implies that `c` is the constant.

If `x` is an mvp object, `constant(x)` returns the value of the constant in the multivariate polynomial; if `x` is numeric, it returns a constant multivariate polynomial with value `x`.

Author(s)

Robin K. S. Hankin

Examples

```
a <- rmvp(5)+4
constant(a)
constant(a) <- 33
a

constant(0)  # the zero mvp
```

`deriv` *Differentiation of mvp objects*

Description

Differentiation of mvp objects

Usage

```
## S3 method for class 'mvp'
deriv(expr, v, ...)
## S3 method for class 'mvp'
aderiv(expr, ...)
```

Arguments

`expr`	mvp object
`v`	Character vector. Elements denote variables to differentiate with respect to
`...`	Further arguments, ignored in `deriv()` but specifies the differentials in `aderiv()`

Details

Function `deriv(S,v)` returns $\frac{\partial^r S}{\partial v_1 \partial v_2 \ldots \partial v_r}$.

Function `aderiv()` uses the ellipsis construction with the names of the argument being the variable to be differentiated with respect to. Thus `aderiv(S,x=1,y=2)` returns $\frac{\partial^3 S}{\partial x \partial y^2}$.

Value

Returns its argument invisibly

Author(s)

Robin K. S. Hankin

Examples

```
p <- rmvp(10,9,9,letters[1:4])
deriv(p,letters[1:3])
deriv(p,rev(letters[1:3]))  # should be the same

aderiv(p,a=1,b=2,c=1)

## verify the chain rule:
x <- rmvp(7,symbols=6)
v <- allvars(x)[1]
s <- as.mvp("1  +  y  -  y^2 zz  +  y^3 z^2")
LHS <- subsmvp(deriv(x,v)*deriv(s,"y"),v,s)   # dx/ds*ds/dy
RHS <- deriv(subsmvp(x,v,s),"y")              # dx/dy

LHS - RHS # should be zero
```

---

drop *Drop empty variables*

Description

Convert a mvp object which is a pure constant into a scalar whose value is the coefficient of the empty term

Usage

```
drop(S)
```

Arguments

S	An mvp object

Author(s)

Robin K. S. Hankin

See Also

subs

Examples

```
m1 <- as.mvp("1+bish +bash^2 + bosh^3")
m2 <- as.mvp("bish +bash^2 + bosh^3")

m1-m2         # an mvp object
drop(m1-m2)   # numeric
```

invert *Replace symbols with their reciprocals*

Description

Given an mvp object, replace one or more symbols with their reciprocals

Usage

```
invert(p, v)
```

Arguments

p	Object (coerced to) mvp form
v	Character vector of symbols to be replaced with their reciprocal; missing interpreted as replace all symbols

Author(s)

Robin K. S. Hankin

See Also

subs

Examples

```
invert("x")

invert(rmvp(10,7,7,letters[1:3]),"a")
```

kahle — *A sparse multivariate polynomial*

Description

A sparse multivariate polynomial inspired by Kahle (2013)

Usage

```
kahle(n = 26, r = 1, p = 1, coeffs = 1, symbols = letters)
```

Arguments

n	Number of different symbols to use
r	Number of symbols in a single term
p	Power of each symbol in each terms
coeffs	Coefficients of the terms
symbols	Alphabet of symbols

Author(s)

Robin K. S. Hankin

References

David Kahle 2013. “**mpoly**: multivariate polynomials in R”. *R Journal*, volume 5/1.

See Also

special

Examples

```
kahle()  # a+b+...+z
kahle(r=2,p=1:2)  # Kahle's original example

## example where mvp runs faster than spray (mvp does not need a 200x200 matrix):
k <- kahle(200,r=3,p=1:3,symbols=paste("x",sprintf("%02d",1:200),sep=""))
system.time(ignore <- k^2)
#system.time(ignore <- mvp_to_spray(k)^2)   # needs spray package loaded
```

`knight` *Chess knight*

Description

Generating function for a chess knight on an infinite $d$-dimensional chessboard

Usage

```
knight(d, can_stay_still = FALSE)
```

Arguments

`d` Dimension of the board

`can_stay_still` Boolean, with default `FALSE` meaning that the knight is obliged to move and `FALSE` meaning that it has the option of remaining on its square

Note

The function is a slight modification of `spray::knight()`.

Author(s)

Robin K. S. Hankin

Examples

```
knight(2)      # regular chess knight on a regular chess board
knight(2,TRUE) # regular chess knight that can stay still

# Q: how many ways are there for a 4D knight to return to its starting
# square after four moves?

# A:
constant(knight(4)^4)

# Q ...and how many ways in four moves or fewer?

# A1:
constant(knight(4,TRUE)^4)

# A2:
constant((1+knight(4))^4)
```

`lowlevel` *Low level functions*

Description

Various low-level functions that call the C routines

Usage

```
mvp_substitute(allnames,allpowers,coefficients,v,values)
mvp_substitute_mvp(allnames1, allpowers1, coefficients1, allnames2, allpowers2,
    coefficients2, v)
mvp_prod(allnames1,allpowers1,coefficients1,allnames2,allpowers2,coefficients2)
mvp_add(allnames1, allpowers1, coefficients1, allnames2, allpowers2,coefficients2)
simplify(allnames,allpowers,coefficients)
mvp_deriv(allnames, allpowers, coefficients, v)
mvp_power(allnames, allpowers, coefficients, n)
```

Arguments

`allnames,allpowers,coefficients,allnames1,allpowers1,coefficients1,allnames2,allpowers2,coeffici`
Variables sent to the C routines

Details

These functions call the functions defined in `RcppExports.R`

Note

These functions are not intended for the end-user. Use the syntatic sugar (as in `a+b` or `a*b` or `a^n`), or functions like `mvp_plus_mvp()`, which are more user-friendly

Author(s)

Robin K. S. Hankin

`mpoly` *Conversion to and from mpoly form*

Description

The **mpoly** package by David Kahle provides similar functionality to this package, and the functions documented here convert between mpoly and mvp objects. The mvp package uses `mpoly::mp()` to convert character strings to mvp objects.

Usage

```
mpoly_to_mvp(m)
## S3 method for class 'mvp'
as.mpoly(x,...)
```

Arguments

`m`	object of class mvp
`x`	object of class mpoly
`...`	further arguments, currently ignored

Author(s)

Robin K. S. Hankin

See Also

`spray`

Examples

```
x <- rmvp(5)

x == mpoly_to_mvp(mpoly::as.mpoly(x))        # should be TRUE
```

`mvp` *Multivariate polynomials, mvp objects*

Description

Create, test for, an coerce to, mvp objects

Usage

```
mvp(vars, powers, coeffs)
is_ok_mvp(vars,powers,coeffs)
is.mvp(x)
as.mvp(x,...)
```

Arguments

`vars`	List of variables comprising each term of a mvp object
`powers`	List of powers corresponding to the variables of the `vars` argument
`coeffs`	Numeric vector corresponding to the coefficients to each element of the `var` and `powers` lists
`x`	Object possibly of class `mvp`
`...`	Further arguments, passed to the methods

Details

Function `mvp()` is the formal creation mechanism for `mvp` objects. However, it is not very user-friendly; it is better to use `as.mvp()` in day-to-day use.

Function `is_ok_mvp()` checks for consistency of its arguments.

Author(s)

Robin K. S. Hankin

Examples

```
mvp(list("x" , c("x","y"), "a",c("y","x")),list(1,1:2,3,c(-1,4)),1:4)

## Note how the terms appear in an arbitrary order, as do
## the symbols within a term.


kahle  <- mvp(
    vars   = split(cbind(letters,letters[c(26,1:25)]),rep(seq_len(26),each=2)),
    powers = rep(list(1:2),26),
    coeffs = 1:26
)

## again note arbitrary order of terms and symbols within a term
```

Ops.mvp *Arithmetic Ops Group Methods for mvp objects*

Description

Allows arithmetic operators to be used for multivariate polynomials such as addition, multiplication, integer powers, etc.

Usage

```
## S3 method for class 'mvp'
Ops(e1, e2)
mvp_negative(S)
mvp_times_mvp(S1,S2)
mvp_times_scalar(S,x)
mvp_plus_mvp(S1,S2)
mvp_plus_numeric(S,x)
mvp_eq_mvp(S1,S2)
```

Arguments

e1,e2,S,S1,S2	Objects of class "mvp"
x	Scalar, length one numeric vector

Details

The function Ops.mvp() passes unary and binary arithmetic operators "+", "-", "*" and "^" to the appropriate specialist function.

The most interesting operator is "*", which is passed to mvp_times_mvp(). I guess "+" is quite interesting too.

Value

The high-level functions documented here return an object of mvp, the low-level functions documented at lowlevel.Rd return lists. But don't use the low-level functions.

Author(s)

Robin K. S. Hankin

See Also

lowlevel

Examples

```
p1 <- rmvp(3)
p2 <- rmvp(3)

p1*p2

p1+p2

p1^3


p1*(p1+p2) == p1^2+p1*p2  # should be TRUE
```

---

print *Print methods for mvp objects*

Description

Print methods for mvp objects: to print, an mvp object is coerced to mpoly form and the mpoly print method used.

Usage

```
## S3 method for class 'mvp'
print(x, ...)
```

Arguments

x	mvp object, coerced to mpoly form
...	Further arguments

Value

Returns its argument invisibly

Author(s)

Robin K. S. Hankin

Examples

```
a <- rmvp(4)
a
print(a)
print(a,stars=TRUE)
print(a,varorder=rev(letters))
```

---

rmvp *Random multivariate polynomials*

Description

Random multivariate polynomials, intended as quick “get you going” examples of mvp objects

Usage

```
rmvp(n, size = 6, pow = 6, symbols = 6)
```

Arguments

n	Number of terms to generate
size	Maximum number of symbols in each term
pow	Maximum power of each symbol
symbols	Symbols to use; if numeric, interpret as the first `symbols` letters of the alphabet

Details

What you see is what you get, basically.

Value

Returns a multivariate polynomial, an object of class `mvp`

Author(s)

Robin K. S. Hankin

Examples

```
rmvp(5)
rmvp(5,symbols=state.abb)
```

special — *Various functions to create simple multivariate polynomials*

Description

Various functions to create simple mvp objects such as single-term, homogenous, and constant multivariate polynomials.

Usage

```
product(v,symbols=letters)
homog(d,power=1,symbols=letters)
linear(x,power=1,symbols=letters)
xyz(n,symbols=letters)
numeric_to_mvp(x)
```

Arguments

`d,n`	An integer; generally, the dimension or arity of the resulting mvp
`v,power`	Integer vector of powers
`x`	Numeric vector of coefficients
`symbols`	Character vector for the symbols

Value

All functions documented here return a mvp object

Note

The functions here are related to their equivalents in the multipol and spray packages, but are not exactly the same.

Function `constant()` is documented at `constant.Rd`, but is listed below for convenience.

Author(s)

Robin K. S. Hankin

See Also

`constant`, `zero`

Examples

```
product(1:3)       #      a * b^2 * c^3
homog(3)           #      a + b + c
homog(3,2)         #      a^2  + a b + a c + b^2 + b c + c^2
linear(1:3)        #      1*a + 2*b + 3*c
constant(5)        #      5
xyz(5)             #      a*b*c*d*e
```

spray *Spray functionality*

Description

Convert between spray and mvp form

Usage

```
spray_to_mvp(L, symbols = letters)
mvp_to_spray(S)
```

Arguments

`L`	mvp object
`symbols`	character vector of symbols
`S`	Spray object

Author(s)

Robin K. S. Hankin

Examples

```
mvp_to_spray(rmvp(5))
spray_to_mvp(spray::spray(diag(6),1:6))
```

subs *Substitution*

Description

Substitute symbols in an mvp object for numbers or other multivariate polynomials

Usage

```
subs(S, ..., drop = TRUE)
subsy(S, ..., drop = TRUE)
subsmvp(S,v,X)
```

Arguments

`S,X`	Multivariate polynomials
`...`	named arguments corresponding to variables to substitute
`drop`	Boolean with default `TRUE` meaning to return a scalar (the constant) in place of a constant mvp object
`v`	A string corresponding to the variable to substitute

Details

Function subs() uses a natural R idiom to substitute scalar values for symbols.

Observe that this type of substitution is sensitive to order:

```
> p <- as.mvp("a b^2")
> subs(p,a="b",b="x")
mvp object algebraically equal to
x^3
> subs(p,b="x",a="b")
mvp object algebraically equal to
b x^2
```

Functions subsy() and subsmpv() are lower-level functions, not really intended for the end-user. Function subsy() substitutes variables for numeric values (order matters if a variable is substitued more than once). Function subsmpv() takes a mvp object and substitutes another mvp object for a specific symbol.

Value

Return a multivariate polynomial, object of class mvp

Author(s)

Robin K. S. Hankin

See Also

drop

Examples

```
p <- rmvp(6,2,2,letters[1:3])
p
subs(p,a=1)
subs(p,a=1,b=2)

subs(p,a="1+b x^3",b="1-y")
subs(p,a=1,b=2,c=3,drop=FALSE)

do.call(subs,c(list(as.mvp("z")),rep(c(z="C+z^2"),5)))
```

zero *The zero polynomial*

Description

Test for a polynomial being zero

Usage

```
is.zero(x)
```

Arguments

x Object of class mvp

Details

Function `is.zero()` returns `TRUE` if `x` is indeed the zero polynomial. It is defined as `length(vars(x))==0` for reasons of efficiency, but conceptually it returns `x==constant(0)`.

(Use `constant(0)` to create the zero polynomial).

Note

I would have expected the zero polynomial to be problematic (cf the **freegroup** and **permutations** packages, where similar issues require extensive special case treatment). But it seems to work fine, which is a testament to the robust coding in the STL.

A general mvp object is something like

`{"x" -> 3, "y" -> 5} -> 6, {"x" -> 1, "z" -> 8} -> -7}`

which would be $6x^3y^5 - 7xz^8$.

The zero polynomial is just `{}`. Neat, eh?

Author(s)

Robin K. S. Hankin

See Also

`constant`

Examples

```
constant(0)

t1 <- as.mvp("x+y")
t2 <- as.mvp("x-y")

stopifnot(is.zero(t1*t2-as.mvp("x^2-y^2")))
```

Index