The SciDAVis Handbook

Ion Vasilief, Roger Gadiou, Knut Franke, Fellype Nascimento, and Miquel Garriga

June 21, 2018

Contents

1 Introduction **1**
1.1 What is SciDAVis? . . . 1
1.2 Command Line Parameters . . . 2
1.2.1 Specify a File . . . 2
1.2.2 Command Line Options . . . 3
1.3 General Concepts and Terms . . . 3
1.3.1 Tables . . . 5
1.3.2 Matrix . . . 7
1.3.3 Plot Window . . . 8
1.3.4 Note . . . 8
1.3.5 Log Window . . . 9
1.3.6 The Project Explorer . . . 10

2 Drawing plots with SciDAVis **11**
2.1 2D X-Y plots . . . 11
2.1.1 2D plot from data. . . . 11
2.1.2 2D plot from function. . . . 15
2.1.2.1 Direct plot of a function. . . . 15
2.1.2.2 Filling of a table with the values of a function. . . . 17
2.1.3 The different types of 2D X-Y plots . . . 18
2.1.4 Customization of a 2D plot . . . 18
2.1.4.1 "Plot details" window . . . 19
2.1.4.1.1 Options for the layer . . . 19
2.1.4.1.2 Custom curves for data series . . . 20
2.1.5 Changing default 2D plot options . . . 22
2.1.5.1 Modification of default options . . . 23
2.1.6 Working with templates . . . 24
2.2 Other special 2D plots . . . 25
2.2.1 Pie plots . . . 25
2.2.1.1 Formatting of pie plots . . . 25
2.2.2 Vectors plots . . . 26
2.2.2.1 Formatting of vector plots . . . 27
2.3 Statistical plots . . . 28
2.3.1 Box plots . . . 28

2.3.1.1 Description of box plots . . . 28
2.3.1.2 Customization of box plots . . . 29
2.3.2 Histograms . . . 30
2.3.2.1 Building of an histogram . . . 30
2.3.2.2 customization of histograms . . . 31
2.4 3D plots . . . 32
2.4.1 Direct 3D plot from a function . . . 33
2.4.2 3D plot from a matrix . . . 34
2.4.3 Customization of a 3D plot . . . 35
2.4.3.1 Modification of color schemes . . . 37
2.4.4 Changing default 3D plot options . . . 37
2.5 Multilayer Plots . . . 38
2.5.1 Building a multilayer plot panel . . . 38
2.5.2 Building a multilayer plot step by step . . . 39
2.6 Adding objects to a plot . . . 41
2.6.1 Adding a text label . . . 41

3 Analysis of data and curves 43
3.1 Fast Fourier Transform . . . 43
3.2 Filtering of data curves . . . 44
3.2.1 FFT low pass filter . . . 45
3.2.2 FFT high pass filter . . . 45
3.2.3 FFT band pass filter . . . 46
3.2.4 FFT block band filter . . . 47
3.3 Correlation and autocorrelation . . . 47
3.4 Convolution of functions . . . 49
3.5 Deconvolution . . . 49
3.6 Fitting of data and curves . . . 49
3.6.1 Non Linear Curve Fit . . . 49
3.6.2 Fitting to specific curves . . . 51
3.6.2.1 Fitting to a line . . . 51
3.6.2.2 Fitting to a polynomial . . . 52
3.6.2.3 Fitting to a Bolzmann function . . . 52
3.6.2.4 Fitting to a Gauss function . . . 53
3.6.2.5 Fitting to a Lorentz function . . . 53
3.6.3 Multi-Peaks fitting . . . 54
3.6.4 Changing default parameters for fitting . . . 55
3.7 Interpolation . . . 56

4 Scripting 57
4.1 muParser . . . 57
4.2 Python . . . 57
4.2.1 Getting Started . . . 57
4.2.2 Python Basics . . . 61
4.2.2.1 Expressions . . . 61
4.2.2.2 Statements . . . 61

4.2.3 Evaluation Reloaded . . . 63
4.2.4 Mathematical Functions . . . 64
4.2.5 Accessing SciDAVis's functions from Python . . . 64
4.2.5.1 Establishing contact . . . 64
4.2.5.2 Working with Tables . . . 66
4.2.5.3 Working with Matrices . . . 67
4.2.5.4 Plotting and Working with Graphs . . . 67
4.2.5.5 Fitting . . . 70
4.2.6 API documentation . . . 71
4.2.6.1 class AbstractAspect (inherits QObject) . . . 71
4.2.6.2 class Column (inherits AbstractAspect) . . . 72
4.2.6.3 class MDIWindow (inherits QWidget) . . . 74
4.2.6.4 class Table (inherits MDIWindow) . . . 75
4.2.6.5 class Matrix (inherits MDIWindow) . . . 77
4.2.6.6 class ArrowMarker . . . 78
4.2.6.7 class ImageMarker . . . 79
4.2.6.8 class Legend . . . 79
4.2.6.9 class QwtSymbol . . . 80
4.2.6.10 class QwtPlotCurve . . . 82
4.2.6.11 class Grid . . . 84
4.2.6.12 class Layer (inherits QWidget) . . . 85
4.2.6.13 class Graph (inherits MDIWindow) . . . 90
4.2.6.14 class Note (inherits MDIWidget) . . . 92
4.2.6.15 class ApplicationWindow (inherits QMainWindow) 92
4.2.6.16 class Fit (inherits QObject) . . . 95
4.2.6.17 class Folder (inherits QObject) . . . 97
4.2.7 The Initialization File . . . 98
4.2.7.1 Recommended approach to per-user configuration . 98

5 Command Reference **99**
5.1 The File Menu . . . 99
5.2 The Edit Menu . . . 107
5.3 The View Menu . . . 110
5.4 The Graph Menu . . . 112
5.5 The Plot Menu . . . 119
5.6 The Plot 3D menu . . . 125
5.7 The Tools Menu . . . 128
5.8 The Analysis Menu . . . 129
5.8.1 Commands for the analysis of data in tables . . . 129
5.8.2 Commands for the analysis of curves in plots . . . 130
5.9 The Table Menu . . . 135
5.10 The Matrix Menu . . . 138
5.11 The Format Menu . . . 139
5.12 The Window Menu . . . 144
5.13 The Help Menu . . . 144

6 The Toolbars **146**
6.1 The File Toolbar . . . 146
6.2 The Edit Toolbar . . . 148
6.3 The Plot Toolbar. . . . 148
6.4 The Graph Toolbar. . . . 150
6.5 The Table Toolbar. . . . 151
6.6 The matrix plot Toolbar. . . . 151
6.7 The 3D Surfaces Toolbar. . . . 151

A Appendix **154**
A.1 Credits and License . . . 154
A.1.1 GNU Free Documentation License . . . 154
A.1.1.1 Preamble . . . 155
A.1.1.2 Applicability And Definitions . . . 155
A.1.1.3 Verbatim Copying . . . 156
A.1.1.4 Copying In Quantity . . . 156
A.1.1.5 Modifications . . . 157
A.1.1.6 Combining Documents . . . 158
A.1.1.7 Collections Of Documents . . . 159
A.1.1.8 Aggregation With Independent Works . . . 159
A.1.1.9 Translation . . . 159
A.1.1.10 Termination . . . 159
A.1.1.11 Future Revisions Of This License . . . 160
A.2 How to obtain SciDAVis . . . 160
A.3 Requirements . . . 160
A.4 Installation from binary packages . . . 161
A.5 Compilation and Installation from sources . . . 161

B Frequently asked questions **162**

List of Figures

1.1 A typical SciDAVis session 3
1.2 A SciDAVis table with the properties dialog developped and the type tag selected. 5
1.3 The two other tags of the properties dialog of SciDAVis tables. 6
1.4 The SciDAVis matrix 7
1.5 An example of SciDAVis 2D graph with 2 layers. 8
1.6 The SciDAVis Note Window 9
1.7 The SciDAVis Note Window used as a calculator 9
1.8 The SciDAVis Log window with the information related to a fit on a curve 10
1.9 The SciDAVis Project Explorer 10

2.1 A simple 2D plot: the table. 12
2.2 A simple 2D plot: the default plot. 13
2.3 A simple 2D plot: the plot finished. 13
2.4 A table with two series of values (X1,Y1) and (X2,Y2). 14
2.5 A 2D plot with two Y and two X axis. 15
2.6 Direct plot of a function. 16
2.7 Direct plot of a parametric function. 16
2.8 Direct plot of a function in polar coordinates. 17
2.9 Function plot: filling of the X column. 17
2.10 Function plot: filling of the Y column. 18
2.11 The *Plot details* Dialog: general properties of the layers. 20
2.12 The *Plot details* Dialog: Plot Associations. 20
2.13 The *Plot details* Dialog: Choice of axes. 21
2.14 The Plot details Dialog: Line formatting. 21
2.15 The Plot details Dialog: Symbol formatting. 21
2.16 The Plot details Dialog: Pattern formatting for bars. 22
2.17 The Plot details Dialog: Spacing formatting for bars. 22
2.18 The preferences dialog: 2D plot options. 23
2.19 An example of pie plot. 25
2.20 Pie segment formatting. 26
2.21 An example of a vector plot (fluid flow around a cylinder in a laminar mode). 27
2.22 Vector-XYXY formatting. 27

2.23 Vector-XYAM formatting. 28
2.24 An example of a box plot for three columns. 29
2.25 The Custom Curves Dialog for box: pattern formatting. 29
2.26 The Custom Curves Dialog for box: whiskers formatting. 30
2.27 The Custom Curves Dialog for box: percentile formatting. 30
2.28 An example of histogram. 31
2.29 Pattern formatting in histograms. 31
2.30 Whiskers formatting in histograms. 32
2.31 Interval selection in histograms. 32
2.32 Example of a 3D Plots. 33
2.33 Definition of a new surface 3D plot 33
2.34 The 3D surface plot created by default 34
2.35 Assigning a multi-lines formula to a matrix 35
2.36 The Contour curves options dialog. 35
2.37 The preferences dialog: 3D plot options. 38
2.38 The text options dialog. 41

3.1 A signal and the FFT dialog box for a plot. 43
3.2 The resulting FFT with the characteristic frequencies. 44
3.3 The **FFT** dialog box for a table. 44
3.4 Signal after a FFT low pass filter 45
3.5 Signal after a FFT high pass filter 46
3.6 Signal after a FFT band pass filter 46
3.7 Signal after a FFT block band filter 47
3.8 An example of a correlation between two functions: the two signals. . 48
3.9 An example of a correlation between two functions: the correlation function. 48
3.10 The first step of the **Fit Wizard** dialog box. 50
3.11 The second step of the **Fit Wizard** dialog box. 50
3.12 The results of the **Fit Wizard**. 51
3.13 The results of a **Quick Fit→Fit Linear**. 52
3.14 The results of a **Quick Fit→Fit Bolzmann (Sigmoïdal)**. 53
3.15 The results of a **Quick Fit→Fit Gaussian**. 53
3.16 The results of a **Quick Fit→Fit Lorentzian**. 54
3.17 The selection of the position of the peaks. 54
3.18 The results of a **Quick Fit→Fit Multipeak→Gaussian**. 55
3.19 The preference dialog for fitting. 55
3.20 Comparison of the three methods of interpolation 56

5.1 The **New→New Function Plot** dialog box. 101
5.2 The **New→New 3D Surface Plot** dialog box. 101
5.3 The **New→New 3D Surface Plot** dialog box. 102
5.4 The result of **Open image File**. 102
5.5 The **Export Graph→Current** dialog. 104
5.6 The basic **Print** dialog. 105
5.7 The **Export Ascii** dialog. 106

5.8 The **Import Ascii** dialog. 107
5.9 The general options dialog: application options. 109
5.10 The plot wizard dialog box. 111
5.11 The project explorer panel. 111
5.12 The undo-redo history. 112
5.13 The scripting console. 112
5.14 The **Add/Remove Curve** dialog box. 113
5.15 The **Add Error Bars** dialog. 113
5.16 A plot with X and Y Error Bars. 114
5.17 The **Add Function** dialog box: cartesian coordinates. 114
5.18 The **Add Function** dialog box: parametric coordinates. 115
5.19 The **Add Function** dialog box: polar coordinates. 115
5.20 The **Add Text** dialog box. 116
5.21 The *Arrow options* dialog: first tab 116
5.22 The *Arrow options* dialog: second tab 117
5.23 The *Geometry* dialog: third tab 117
5.24 The **Add Layer** dialog box. 118
5.25 The **Arrange Layers** dialog 118
5.26 The **Integrate** dialog box. 131
5.27 The **Smooth→Savitsky-Golay** dialog. 132
5.28 The **Smooth→Moving Window Average** dialog. 132
5.29 Comparison of the two smoothing methods. 133
5.30 The dialog and an example of FFT smoothing. 133
5.31 2D plot options dialog: General settings. 140
5.32 Surface plot options: general settings. 141
5.33 Plot options dialog: scales settings. 141
5.34 Surface plot options: scales settings. 142
5.35 General plot options dialog: the axis tab. 142
5.36 General plot options dialog: the grid tab. 143
5.37 Surface plot options dialog: the title tab. 144

6.1 The SciDAVis File Toolbar 146
6.2 The SciDAVis Edit Toolbar 148
6.3 The SciDAVis Plot Toolbar with its different sub-menus 148
6.4 The SciDAVis Graph Toolbar with its different sub-menus 150
6.5 The SciDAVis Table Toolbar 151
6.6 The SciDAVis matrix Plot Toolbar 151
6.7 The SciDAVis 3D Surfaces Toolbar 151

List of Tables

4.1 Supported Mathematical Operators 58
4.2 Mathematical Functions 59
4.3 Non-Mathematical Functions 60
4.4 Supported Mathematical Functions 65

6.1 File toolbar commands. 147
6.2 Edit toolbar commands. 148
6.3 Plot toolbar commands 149
6.4 Plot toolbar commands 150
6.5 Table toolbar commands. 151
6.6 3D Plot toolbar commands. 152
6.7 3D Plot toolbar commands. 153

Abstract

This document is a handbook for using SciDAVis, a program for two- and three-dimensional graphical presentation of data sets and for data analysis. It is updated for version 1.22 and newer. This manual is organized in several chapters:

- The first one describes the main concepts and terms which are used in SciDAVis.
- The second and third chapters build a tutorial on how to obtain plots from different data sets, and to perform mathematical and statistical analysis of data and curves. They are the ones you need to read first to understand the basics of SciDAVis and to be able to work with.
- The next chapter describe some specific possibilities of SciDAVis, that is the scripting.
- The two last chapters are descriptions of all the commands and toolbars used in SciDAVis. These chapters are the reference manual of SciDAVis.

Chapter 1

Introduction

1.1 What is SciDAVis?

SciDAVis stands for *Sci*entific *D*ata *A*nalysis and *Vis*ualization. It is a free cross-platform program for two- and three-dimensional graphical presentation of data sets and for data analysis. The plots can be produced from data sets stored in tables, in matrix or from analytical functions.

The SciDAVis project started as a fork of QtiPlot with the aim of introducing some changes in design as well as project structure. The QtiPlot development was initiated in 2004 by Ion Vasilief. He was the only programmer until May 2006 when Knut Franke and Tilman Hoener zu Siederdissen joined the project. Not much later, Roger Gadiou officially joined as the main documentation writer. In June 2007, insuperable disagreements among the developers lead to the fork and the creation of the SciDAVis project by Knut Franke and Tilman Hoener zu Siederdissen, soon followed by Roger Gadiou. In November 2012, after ~two years of inactivity in the project, Russell Standish assumed the development of SciDAVis. The project is hosted partially at Sourceforge (download files, the bug tracker, forums, mailing lists, etc.), but its source code development was moved from the SciDAVis subversion repository to Github in June 2015.

SciDAVis aims to be a tool for analysis and graphical representation of data, allowing powerfull mathematical treatment and visualization of scientific data while keeping a user-friendly graphical user interface. Another keypoint for the SciDAVis project is to be a multi-system software, it should work on Windows, Linux, and OS-X systems.

SciDAVis is a dynamic tool, the plots created from data sets and the spreadsheets owing the data are interconected. When the spreadsheets are modified, all the objects in the depending plots (curves, axes scales, legends) are automatically updated. For example, deleting a spreadsheet or only some columns will automatically remove all the corresponding curves from the depending plots.

All settings of a complete set of tables, matrix and plots can be saved in project files, having the extention ".sciprj". These project files may be opened using the command line, using the File menu, or by using the icon from the File toolbar.

The plots can be exported to several graphic formats such as JPEG or PNG and

inserted as images in documents or presentations.

Data analysis operations (integration, interpolation, FFT, curve fitting, etc) can be performed on the curves in a 2D plot via the Analysis-plots menu. The results of all these operations are also stored in the project files. They can be visualized at any moment using the Results Log command and can be deleted from the project file via the Clear Log Information command.

When the application is launched, a new project file is created consisting of a grey main window (the workspace) which contains an empty table. In order to be operational, this workspace must be populated with tables storing data sets, either by creating empty tables first (New→New Table command) and then filling them with data, or by importing ASCII files (Import Ascii command), which automatically creates new tables.

The user can easily navigate through the objects of a project file using the Project Explorer command or the Windows menu. The project explorer also allows the user to perform various operations on the windows (tables and plots) in the workspace: hiding, minimazing, closing, renaming, printing, etc.

1.2 Command Line Parameters

1.2.1 Specify a File

When starting SciDAVis from the command prompt, you can supply the name of a project file:

```
SciDAVis file_name.sciprj
```

Other file format are also accepted: *.opj, .ogm, .ogw, .ogg* for Origin projects, and *.qti, qti.gz* for Qtiplot projects.

The name can also refer to an ASCII file:

```
SciDAVis ASCII_file_name
```

In this latter case a new "untitled" project will be created, containing a spreadsheet with the ASCII data in the file and a 2D plot of all columns as a function of the first column in the file. You must take care of the format of the ASCII file because it will be read with the current parameters of the Import Ascii command dialog. The default values are:

- the default field separator is ; but it can be changed in the Preferences command dialog,
- all lines are read,
- the first line is used to name the columns,
- the spaces at the end of the lines are not removed,
- the spaces are not simplified.

1.2.2 Command Line Options

Valid options are:

- -a or --about: show about dialog and exit
- -h or --help: show command line options
- -l=XX or --lang=XX: start SciDAVis in language XX ('en', 'fr', 'de', ...)
- -m or --manual: show SciDAVis manual in a standalone window
- -v or --version: print SciDAVis version and release date
- -x or --execute: execute the script file given as argument

1.3 General Concepts and Terms

Several plots and all the data related to these plots can be save in a *project* file, the project is therefore the main container of SciDAVis. The following screenshot gives an example of a typical session. This example shows the log panel at the top of the workspace, the project explorer at the bottom, a table and a plot window are shown while other are docked or hidden.

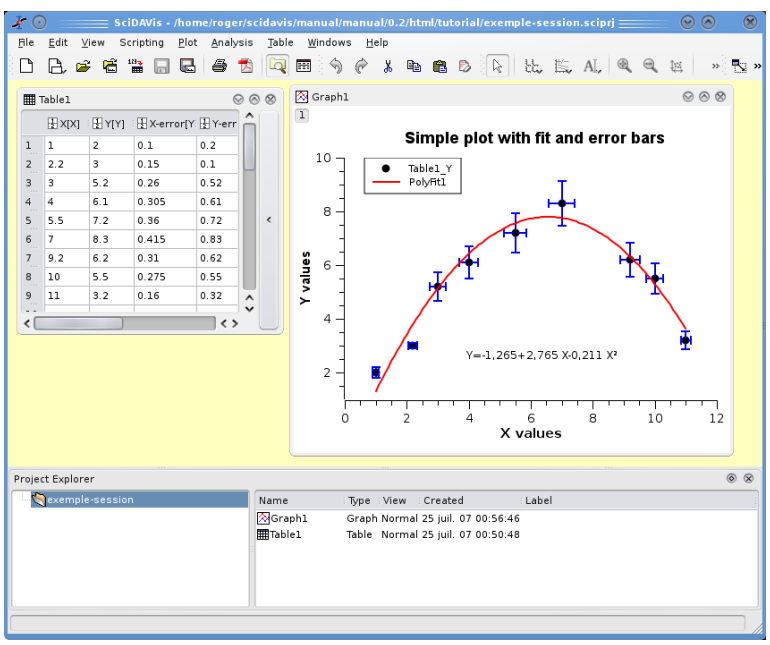


Figure 1.1: A typical SciDAVis session

There are numerous commands available in SciDAVis depending on the element which is selected. Therefore, the main menu bar changes when you select a particular element of the project. Moreover, you can access to the set of commands relevant of an element by activating the context menu with the right button of the mouse.

In a project, the objects which can be used are:

Tables A table is a spreadsheet which can be used to store the datas you are entering. It can also be used to do some calculations and statistical analysis of datas. In each table, columns can be labelled as X-values or Y-values for 2D-plotting, or Z-values if you plan to build a 3D-plot. In addition, columns can be labelled as errors on X or on Y values (see Set Column as command).

A table can be created by the New→New Table command. Then there are several ways to fill the table with your data. If you want to read a table from an ASCII file, you can import the data from the file to a table with the Import Ascii command. You can also enter each value from the keyboard, or copy and paste from another spreadsheet software. The last way to enter your data is to fill the table with the results of a mathematical function (Assign Formula command from the Table menu)

Matrix A matrix is a special table which is used to store the data points for surface 3D plots. It contains Z-values and doesn't include any column or row which could be designed as X-values or Y-values. Nevertheless, you can specify the X-values and the Y-values with the Set Coordinates command command from the Matrix menu.

A matrix can be created by the New→New Matrix command. If you want to read a matrix from an ASCII file, you can import the data of the file to a table with the Import Ascii command and then convert this table to a matrix with the Convert to Matrix command. In the same way as for tables, you can also fill matrix with the results of a function z=f(i,j) in which i and j are row and column numbers, or z=f(x,y). (see Assign Formula command from the Matrix menu)

A Graph A graph can contain one or several plots. Each of these plots is contained in a different *layer*, these layers can be arranged in many ways to build matrix of plots.

A new layer can be added to an existing graph with the Add Layer command from the Graph menu. you can also remove an existing layer with the Remove Layer command, but if you remove a layer, the plot will be deleted. You can also copy a layer from one graph to another, or copy an existing graph into another, the window will be added as a new layer (see the section on Multilayer Plots for more details).

Plots can be created in several ways. You can select data in tables or matrix and build a plot, or create new plots from functions of one or two variables (see sections 2D plots and 3D plots).

A Note This window is a text container which can simply be used to insert comments into a project. This object is nevertheless far more powerfull than that: it can be

used as a calculator, for executing single commands and for writing scripts (see the Scripting section for more details).

The Log Window This window is used to store the results of all the calculations which have been done. If this window is not visible, you can find it with the Project Explorer or with the Results Log command.

The text in the log window is also saved in the project file, so that when you load a previously saved project, the results-log panel is re-filled with the results of the calculations.

The Project Explorer This window is used to list all the windows contained in a projet. The Project Explorer is opened by the Project Explorer command, and gives a quick access to all elements of a project, hidden or visibles. It can be used to do some operations on the windows related to these items such as hidding a window, renaming windows, etc.

A project file can include several independant projects. In this case, the containers of each project are stored in different folders.

1.3.1 Tables

The table is the main part of SciDAVis when working with data. For controlling and converting data the spreadsheet contains a highly customizable table: all colors and font preferences can be set using the Preferences command of the Edit menu. You can resize a table in terms of rows and columns using the Dimensions command command of the Table menu. On the left side of the table, the button can be used to develop the properties tags. This allows to customize the main parameters of the table.

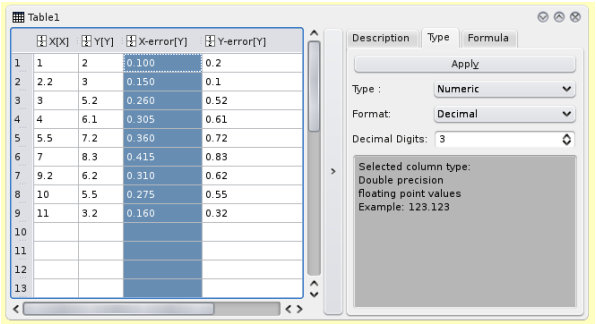


Figure 1.2: A SciDAVis table with the properties dialog developped and the type tag selected.

In a spreadsheet, columns can have the following flags: X, Y, Z, X-error, Y-error or can be simple columns without any special flag. The X columns are abscissae columns while the Y columns are ordinates columns used when creating a 2D plot from data.

The X-error and Y-error columns can be used in order to add error bars to 2D plots. These flags can be changed using the Set Column as command.

The tag which is selected in figure 1.2 is used to assign a type to columns: numeric, text, date or time. The format used to display the data can then be chosen, the format in tables is not used for plots (use the Axes command to define display format for axes labels).

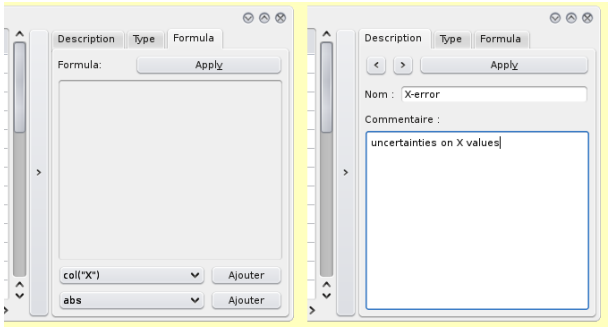


Figure 1.3: The two other tags of the properties dialog of SciDAVis tables.

Every column of the table has a label, this can be defined in the description tag (figure 1.3). This label will be used by default in plots for curve selection and legend display. You can use complex labels with spaces and special characters like "Vol (cc/g)" if needed. This command can also be reached by the Edit Column Description command of the Table menu.

The last tag of the properties dialog correspond to the command Assign Formula command of the Table menu (figure 1.3). It is used to fill the column with the result of a mathematic expression. Refer to the Assign Formula command for more details.

You can select all the columns of the spreadsheet (Ctrl+A) or only some of them by clicking on the column label while keeping the Ctrl key pressed, or by moving the mouse over the column label. This also allows you to deselect columns.

On the selected columns you can perform various operations:

- Fill with data. You can insert the row numbers (Fill Selection With→Row Numbers command), random numbers (Fill Selection With→Random Values command), or the result of a function (Assign Formula command);
- normalize columns with the **Normalize Columns** command of the context menu;
- sort columns with the Sort Table command of the Table menu or with the **sort column** command of the context menu;
- compute statistical data on columns and rows with the Statistics on Columns command and Statistics on Rows command of the Analysis-tables menu;
- build a plot from selected columns with the **plot** command of the context menu or with the commands of the Plot menu.

All these functions can be reached by right clicking when a column is selected. Most of them can also be reached by using the Table menu.

You can cut, copy and paste data between spreadsheets or between a spreadsheet and another application (Excel, Gnumeric, OpenOffice Calc, etc).

You can import single or multiple ASCII files using the Import Ascii command from the File menu. This will create one or more new tables. You can also export the data from the spreadsheet to a text file using the Export Ascii command.

1.3.2 Matrix

The matrix is a special table which is used for data which depends on two variables. This special table is used to store data for 3D-plots. The difference between a table and a matrix is that there is that columns are assigned to the abscissae x while rows define abscissae y.

The defaut size of a matrix is 32x32 cells. You can modify this size with the Dimensions command. Column and row numbers are named i and j respectively, ranging from 1 to the size of the matrix. You can specify an X-scale and an Y-scale with the Set Coordinates command, this define values of x and y for columns and rows (figure 1.4).

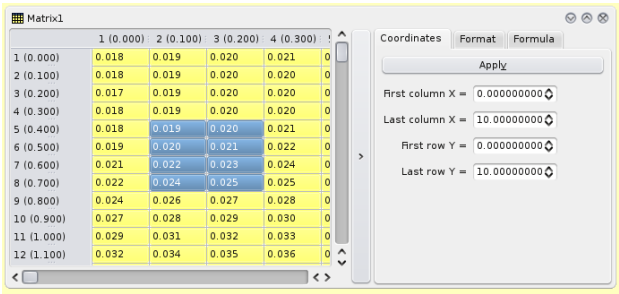


Figure 1.4: The SciDAVis matrix

The values which are stored in a matrix can be obtained from a function of the form z=f(i,j) or z=f(x,y) with the Assign Formula command. They can also be read from a file with the Import Ascii command which allows to read a file in a table, then the table can be converted to a matrix with the Convert to Matrix command of the Matrix menu.

As in the case of tables, a property tag can be shown or hidden by clicking on the vertical button on the right.

Through the Matrix menu, several operations can be done on a matrix like transposition (Transpose command), mirroring (Mirror Horizontally command and Mirror Vertically command), inversion (Invert command), computation of the determinant (Determinant command). The data of a matrix can then be used to build a 3D plot with the commands present in the plot3d menu and in 3D surface toolbar.

1.3.3 Plot Window

The plot window is the one in which the graphic is plotted. The main container of the plot window is the layer. You can have several layers in a plot, which may be arranged as you want. Each layer can contain a plot, or another item like a label.

Each new plot can be inserted in a new layer of this plot window, it has its own geometry and graphic properties (background color, frame, etc). The figure 1.5 shows a graph with two layers which have different geometries. Inside a layer, the area in which the curves are plotted is the *canvas*.

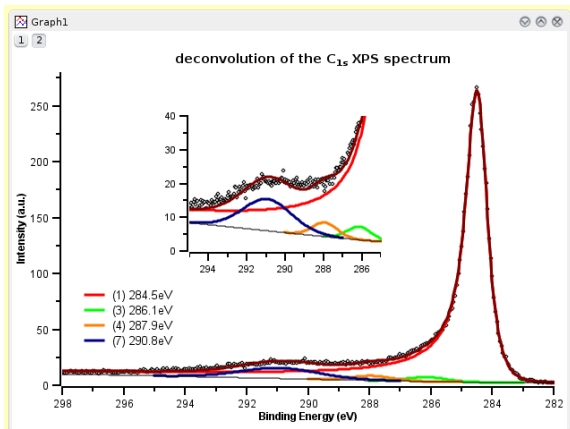


Figure 1.5: An example of SciDAVis 2D graph with 2 layers.

Each layer can be activated by clicking on the corresponding gray button 1 2 in the top-left corner of the window. The elements which can be accessed by a double click in a layer are:

- the graph itself: this will open the Plot details dialog box. You can then change the way the curves are plotted.
- The axes or the axes labels: this will open the General Plot Options Dialog. It is used to customize the axes, the numbers and labels of the axes, and the grid.
- Any other text item: this will open the Text Dialog which allows to customize the font of the label and the frame in which it is drawn.

All these functions can be reached through the Format menu.

1.3.4 Note

A note can simply be used to insert text (comments, notes, etc) into a project, but is really far more powerfull than that. It can be used as a calculator, for executing single commands and for writing scripts.

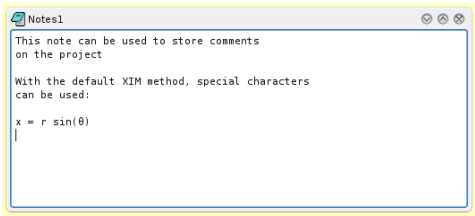


Figure 1.6: The SciDAVis Note Window

You can also change the text input method. *Simple Composing Input Method* is the standard method to enter text in QT applications. *Xim* is the X input method, it is the legacy system of the X window environment to support localized text input. The default choice is the second one, it allows to enter special characters and accents from your localised environment.

The second use of notes is calculator. The evaluation of mathematical expressions and execution of code is done via a note's context menu, the Scripting menu or the convenient keyboard shortcuts. In figure 1.7, it is shown that you just need to place the cursor on an expression and use the Ctrl-Return command to evaluate the expression.

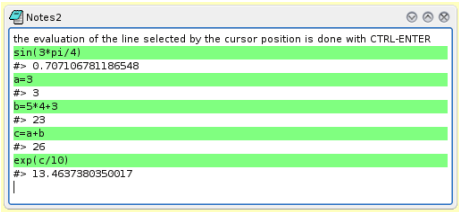


Figure 1.7: The SciDAVis Note Window used as a calculator

you can define variables and refer to them to build complex expressions, but you must evaluate each line with the Ctrl-Return command to fill the variable with its value. All variable are private to the note in which it is defined, and you can't refer to it in another note. With a right click, you access to the context menu which contains the list of all available mathematical functions.

For information on expression syntax, supported mathematical functions and how to write scripts, see the scripting section.

1.3.5 Log Window

This window keeps a history of all analysis which have been done in the project. This panel contains the results of all the correlations, fittings, etc. It can be shown or hidden with the Results Log command of the View menu.

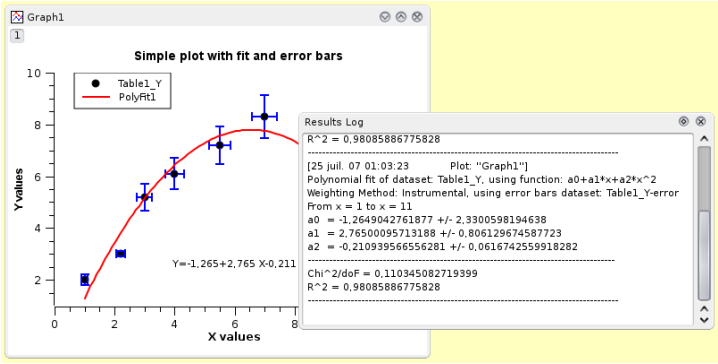


Figure 1.8: The SciDAVis Log window with the information related to a fit on a curve

You can clear the content of the log window with the command Clear Log Information command of the Edit menu. If you load a project for which some analysis has been done, the computations will be done again and the log window will be filled with the results.

1.3.6 The Project Explorer

The project explorer can be opened/closed using the Project Explorer command from the View menu or by clicking on the in the file toolbar.

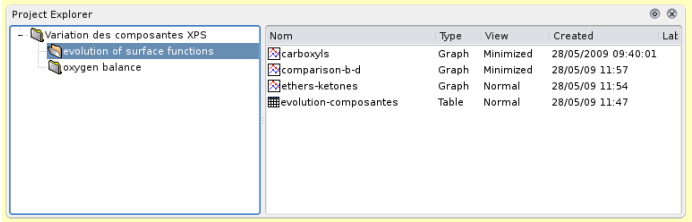


Figure 1.9: The SciDAVis Project Explorer

It gives an overview of the structure of a project and allows the user to perform various operations on the windows (tables and plots) in the workspace: hiding, minimazing, closing, renaming, printing, etc... These functions can be reached via the context menu, by right-clicking on an item in the explorer.

By double-clicking on an item, the corresponding window is shown maximized in the workspace, even if it was hidden before.

You can organize the differents objects in folders. When selecting a folder, the default policy is that only the objects contained in it will be showed in the workspace window. You can also display all the objects in the subfolders if you change this policy with the "View Windows" command to "Windows in Active Folder and Subfolders".

Chapter 2

Drawing plots with SciDAVis

2.1 2D X-Y plots

A 2D plot is based on curves which are defined by Y values as functions of X values. There are different ways to obtain a 2D plot depending on the way the (X,Y) values are defined:

- You can have your (X,Y) values in a table. You need to select at least one column as X values and one column as Y values. This is specified with the Set Column as command. Then you can select the columns and use one command of the Plot menu to plot the data.
- If you want to plot a function, you don't need a table. You can use directly the New→New Function Plot command. This will open the corresponding dialog box and you will be able to define the mathematical expression of your function. In this case, plot can be obtained from functions in cartesian coordinates Y(X), but also in parametered coordinates (X(t),Y(t)), or in angular coordinates r(theta);.
- The combined way is to define a table, and then to fill in the table with the results of functions. This can be done with the Assign Formula command. Then you can select the columns and use one command of the Plot menu to plot the data.

SciDAVis will create a new graph window, and the plot will be inserted in a new layer.

Once the plot is created, you can customize all the graphic items of the plot with the commands of the Format Menu. You can add new items (text labels, lines or arrows, new legend, images) on the plot with the commands of the Graph Menu.

2.1.1 2D plot from data.

The data must be stored in a table. There are several possibilities to insert your (X,Y) values in the table: you can write them directly from the keyboard, or read them from

a file. Here we will use the first solution, refer to the Import Ascii command to use the second one.

The first step is to create an empty project with the New→New Project command from the File menu, you can also use the key CTRL+N or the icon from the File toolbar. Then create a new table with the New→New Table command from the File menu or with the CTRL+T or with the icon from the File toolbar.

At its creation, the table has two column (one for X and one for Y) and 32 rows. You can add rows and columns by selecting a row or a column and using the right button of the mouse, you can also modify the number of rows and columns with the Dimensions command from the Table menu. Then, enter your values, and you obtain the table shown in figure 2.1

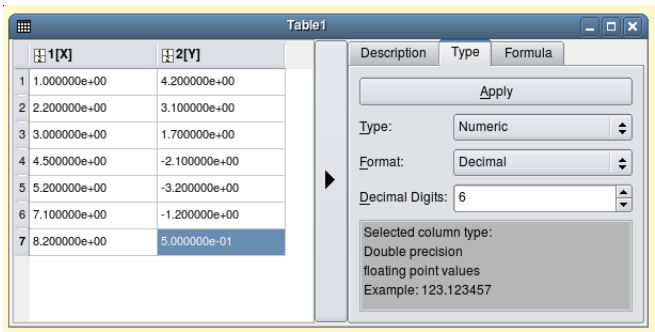


Figure 2.1: A simple 2D plot: the table.

Then, you have to select the two columns, and build your plot (here a simple 2D scatter) with the Scatter command from the context menu, or by clicking on the corresponding icon from the Plot toolbar or with the Scatter command from the Plot menu. A plot is created which uses the default options for all elements. You can customize these default options with the 2D plot preferences dialog. With the default options, you obtain the plot shown in figure 2.2.

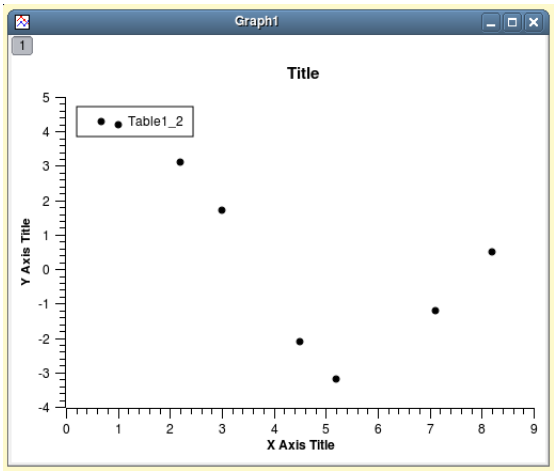


Figure 2.2: A simple 2D plot: the default plot.

You can now customize your plot with the commands of the Format menu. By double clicking on the data points, you open the Plot command dialog which is used to modify the symbols. Then a double-click on the axes opens the Axes command dialog, and you can change the scales, the fonts for the axes labels, etc. You can also add grid lines on X or Y axes (Grid command), etc. Finally, a double click on each text item (X title, Y title, plot title) allows to change the text and the presentation of these elements. See the customize section for more details. An example of the final plot is shown in figure 2.3.

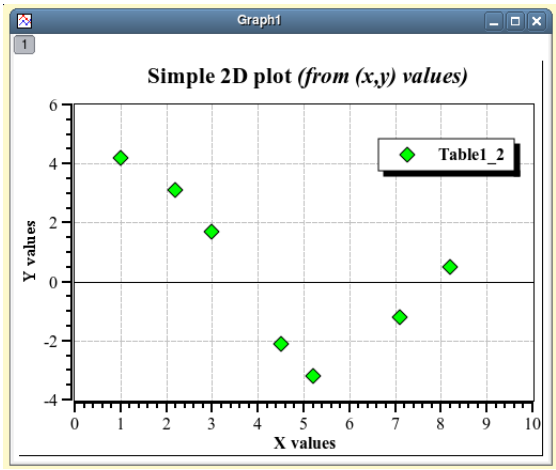


Figure 2.3: A simple 2D plot: the plot finished.

Finally, you have to save your project in a '.sciprj' file with the Save Project command from the File menu or with the CTRL+S or with the icon from the File toolbar. Depending on your application, you can export your plot to a standard image file with the command Export Graph→Current command from the File menu (or with the ALT+G keycode).

There are several types of plots which can be built from a table. They are presented in the Plot menu. One important feature is that it is possible to use up to four axis for the data. For example, create a new table, modify its dimension to 4 columns and 7 rows. Then select the third column and set it as X with the Set Column as command from the Table menu; you can then enter the values of two series (X1,Y1) and (X2,Y2) as show in the figure 2.4.

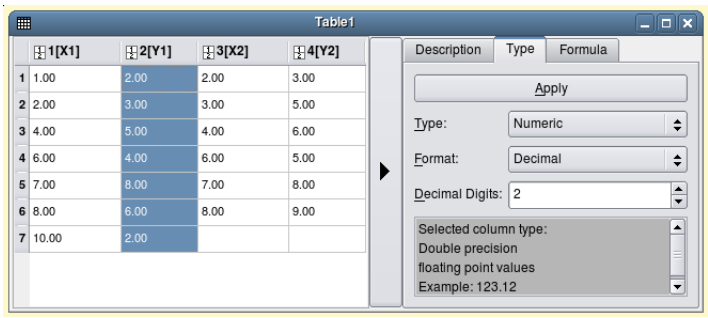


Figure 2.4: A table with two series of values (X1,Y1) and (X2,Y2).

To build the plot, select the two Y column (select Y1, and the select Y2 with CTRL key), use the Plot menu. You obtain a simple plot with two axes, then use the Plot command. In the left window, select the data serie for which you want to change the axes, click on the *axis* tag and define the axes you want to use. After this, the plot is modified but the new axes are not shown. Use the Axes command, select the new axes and click on the *show* checkbox. You can then customize your plot in order to obtain the result presented in figure 2.5.

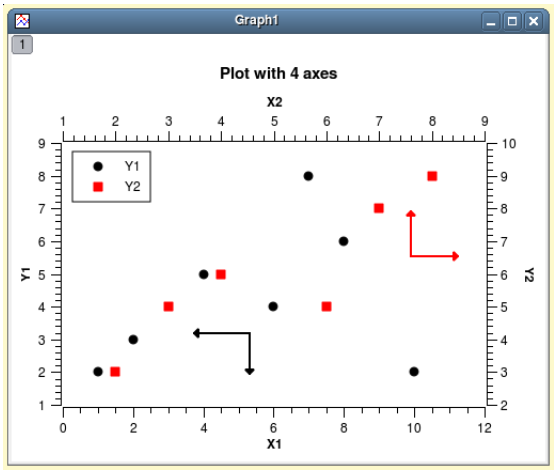


Figure 2.5: A 2D plot with two Y and two X axis.

In addition to the customization which has been already described, four arrows were added with the Draw Arrow command.

2.1.2 2D plot from function.

There are two ways to obtain such a plot: you can plot directly a function or fill a table with the values calculated from this function before doing a plot in the classical way.

2.1.2.1 Direct plot of a function.

If you just want to plot a function, you can use the New→New Function Plot command from the File menu or with the CTRL+F or with the icon from the File toolbar.

You can then enter the expression of your mathematical function, the X range used for the plot, and the number of points used in this X range. See the Add Function command for details.

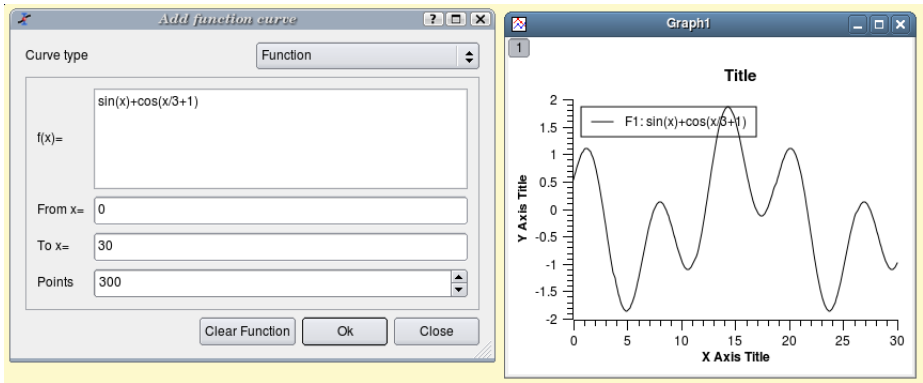


Figure 2.6: Direct plot of a function.

The generic plot which is created by this command can then be customized as explained in the previous section. Beside classical Y=f(X) functions, parametric functions and polar functions can be defined. In parametric coordinates, *X* and *Y* are defined as functions of an independant parameter *m*. You can define these functions, the range for *m* and the number of points computed in this range.

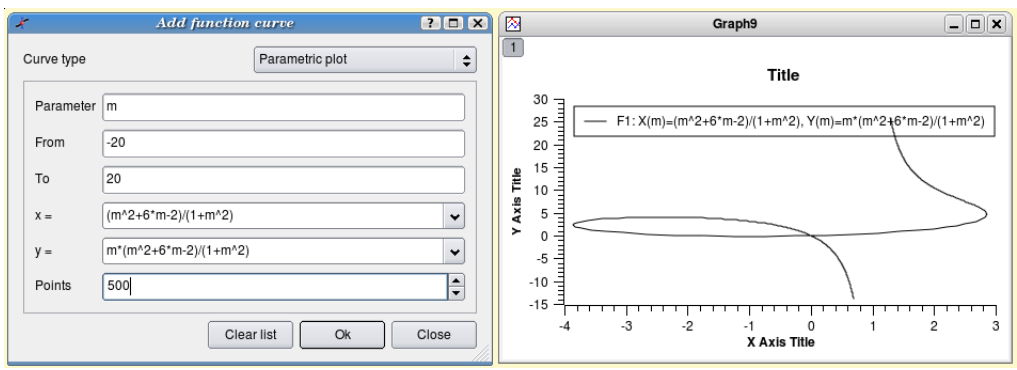


Figure 2.7: Direct plot of a parametric function.

Polar coordinate are defined as a radius *R* and an angle *theta* (in radian). The coordinates are then obtained by *X=R.cos(theta)* and *Y=R.sin(theta)*. You can use a parametric definition: *R=f(t)* and *theta=f(t)*, the range for *t* and the number of points computed in this range.

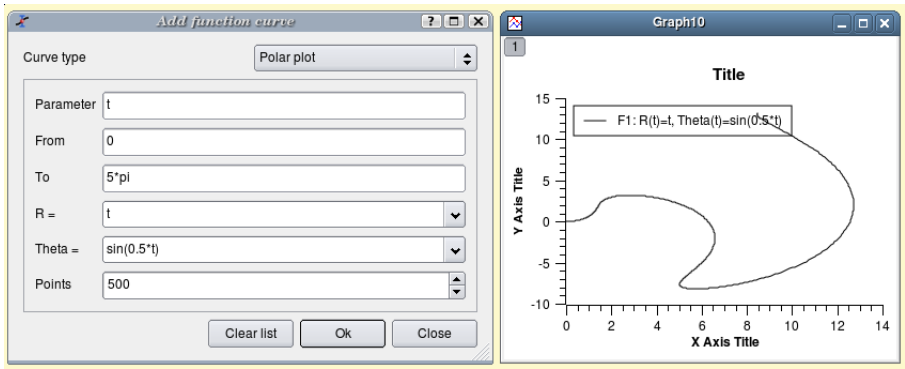


Figure 2.8: Direct plot of a function in polar coordinates.

2.1.2.2 Filling of a table with the values of a function.

If you just want to work not only with the plot but also with the data, you can create a new table as explained in the previous section. Then you can fill this table with the values of a function with the Assign Formula command. The main advantage of this method is that you can do further analysis of the calculated data as the are kept in a table.

To obtain the same plot as in the previous example, you need to create a new table (key CTRL+T) and use the Dimensions command to define 300 rows, then select the first column and use the command Assign Formula command from the context menu, or from the Table menu. The row number symbol is *i*, so you can enter the function expression *i/10* (figure 2.9).

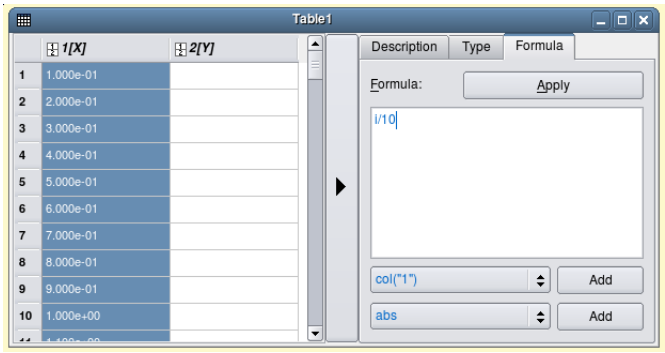


Figure 2.9: Function plot: filling of the X column.

The second step is to select the second column and use the same command. The expression is a function of the X values, that is the first column named *col(1)* (figure 2.10).

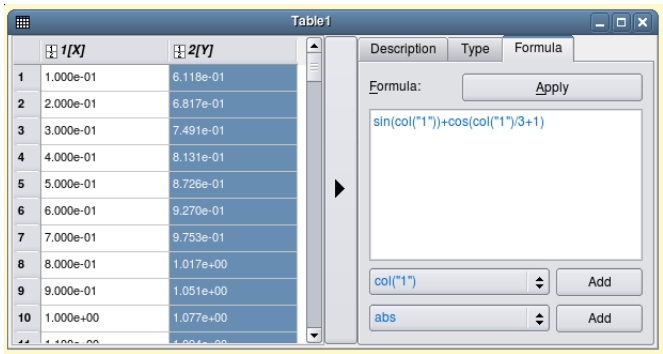


Figure 2.10: Function plot: filling of the Y column.

Once the table is ready, you just have to build the plot as explained in the previous section.

2.1.3 The different types of 2D X-Y plots

Beside the conventional X-Y plots with lines and points, other kinds of plots are available in SciDAVis. Although the presentation of the data can be very different, they are all based on the use of one column for X values and one column for Y values. Follow the links to the corresponding commands to see a description of these plots.

The first set is available in the subset *Special Lines/Symbol* of the Plot menu:

- Drop lines plot (Special Line/Symbol→Vertical Drop Lines command)
- Scatter plot with a smoothed line connection between the points (Special Line/Symbol→Splines command)
- Vertical steps plot (Special Line/Symbol→Vertical Steps command)
- Horizontal steps plot (Special Line/Symbol→Horizontal Steps command)

The other ones are more special plots which can be accessed directly in the Plot menu:

- Vertical bars plot (Vertical Bars command)
- Horizontal bars plot (Horizontal Bars command)
- Area plot (Area command)

2.1.4 Customization of a 2D plot

There are many way to improve and modify the plots:

- The first part of the commands are used to modify the main elements of the plot, that is axis, labels, etc. They can be accessed through the Format menu.

- The second part of commands can be used to insert additional objects likes arrows, images, text labels, etc. They can be accessed in the Graph menu.

This section will show an overview of the first set of commands. See the Graph menu section for the other commands. There are three main windows which allows to modify the plot:

- The Plot command, it is entitled "Plot details" and is used to customize the global properties of the plot (background color, etc) and the data series (points shape, line width, etc).
- The second one is entitled "General plot options" and contain the commands to format axes (scales, labels, grids, etc), it can be accessed through the Scales command, Axes command and Grid command.
- The last one is the Title command which is used to control the properties of the title of the plot.

All these commands are accessed through the Format menu.

2.1.4.1 "Plot details" window

This window has two parts, the left one shows a tree view of the main elements of the plot: the layers and the data series which are plotted in each layer. The main dialog is activated by selecting the Plot command from the Format menu. If it is activated by a double click on a curve in the plot, the same dialog will be opened with the corresponding curve selected (see next section for details). The right section of the window shows the options which are available for the selected entity. If you do some changes, don't forget to click on the *Apply* button before switching to another entity.

2.1.4.1.1 Options for the layer This dialog can be used to modify the background color of the global plot area (i.e. the layer), the color of the canvas (that is the area in which curves are plotted), and the border of the plot. This border is for the global plot, if you want to add a border to the canvas, you can use the *general* tag of the Axes command. If the image format use to save the plots support it, you can also control the transparency of these objects through the opacity parameter. The default value is 255 which means no transparency. See the Export Graph command for details on image formats.

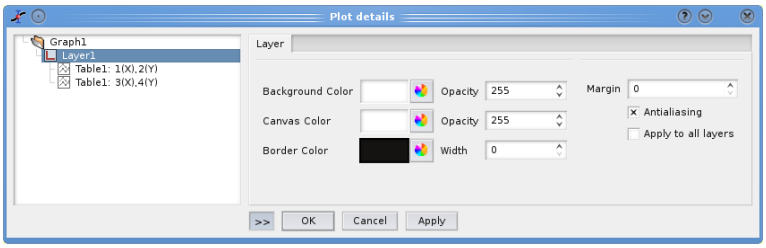


Figure 2.11: The *Plot details* Dialog: general properties of the layers.

2.1.4.1.2 Custom curves for data series The commands can be accessed by a double click on a curve, or by using the Plot command and selecting a curve in the window on the left. The right part of the dialog box contains several tabs which depend on the kind of plot that you are using. The left part of the dialog window shows the curves which are plotted in the active layer. All the modifications will be done on the selected curve.

In this dialog box, beside the customization of data curves, you can change the columns which are used by clicking on the *Plot Associations...* button. This will open a dialog which can be used to select the columns of the table which are used as X and Y values.

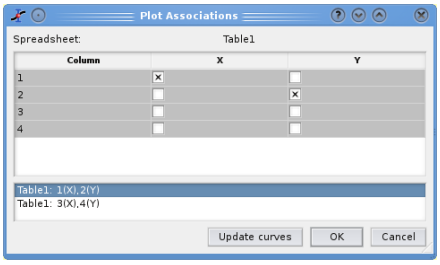


Figure 2.12: The *Plot details* Dialog: Plot Associations.

The button *Worksheet* can be used to access to the table which contains the columns selected.

The dialog presented in figure 2.14 is activated for plots drawn with *symbols*, *line+symbols*, *lines*, *vertical drop lines*, *steps* and *splines*. The first tab labelled *axis* can be used to select the axis which are used for each curve of the plot: bottom (default) or top for abscissae, and left (default) or right for Y values. Beware that whatever your choice the right and top axis will not be drawn, you need to use the Axes command to obtain a plot in which these axis are shown.

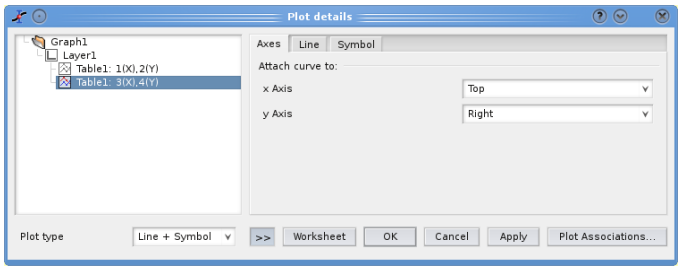


Figure 2.13: The *Plot details* Dialog: Choice of axes.

The second tab allows to modify the style of the line (color, line style, thickness). The connect button allows to change the style which is used to draw the selected curve (steps, droplines, etc). See the Plot menu to see examples of the different types of plot available.

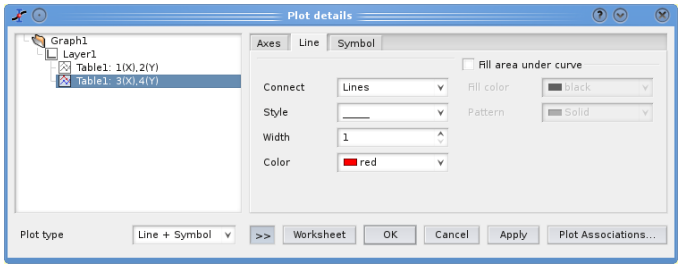


Figure 2.14: The Plot details Dialog: Line formatting.

If you select a style with symbols (scatter or symbol+lines), a last tab can be activated to select the symbol, and to modify the size, the color and the filling color of the symbols.

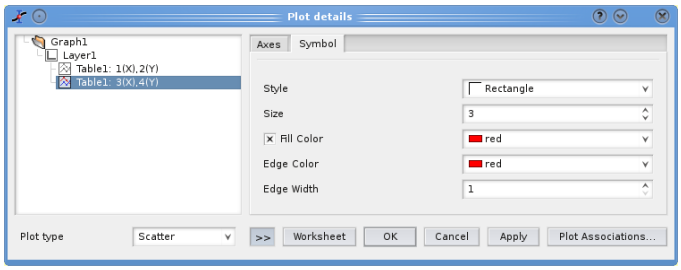


Figure 2.15: The Plot details Dialog: Symbol formatting.

When the data are plotted using bars, the *Plot details* window shows different op-

tions. The first tab named *Pattern* can be used to customize the background and the border lines of the bars.

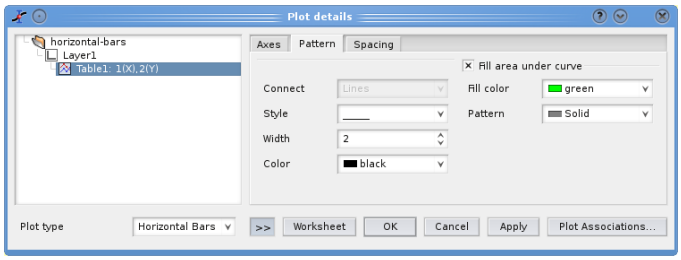


Figure 2.16: The Plot details Dialog: Pattern formatting for bars.

The second tab named *Spacing* can be used to modify the geometry of the bars:

- The default width W of the bar is computed from the smallest difference between two successive abscissae, this correspond to a *Gap between bars* equal to 0 (which is the default value). All bar are drawn with the same width. The Gap is a percentage of this default width: that is, a value of 50 will decrease the width of all the bars by a factor 2.
- The bars are placed in order to be centered around each X value, i.e. between x-W/2 and x+W/2; this correspond to an *Offset* of 0 (default value). The offset is again a percentage of the default width of the bar. For example, a value of 50 will shift the position of the bar by a half of the default width (W/2) and therefore each bar will placed between x and x+W. Negatives values can be used to shift the bars to the left. If inverted axes are used, the direction of the shift remains the same (i.e. positive offset lead to a shift to the right).

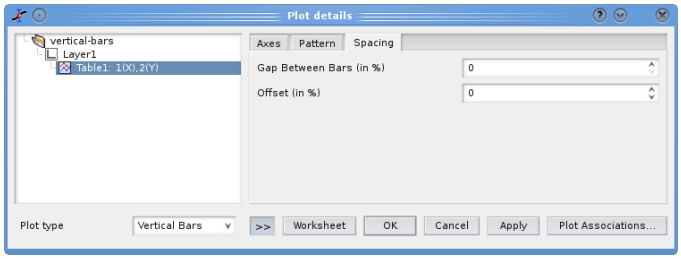


Figure 2.17: The Plot details Dialog: Spacing formatting for bars.

2.1.5 Changing default 2D plot options

There are two ways to modify the default style which is used for plots. The first one applies to all plot and is reached by the Preferences command (in the Edit menu). And

the second one is to define templates for a specific family of plots with the Save as Template command.

2.1.5.1 Modification of default options

In the dialog box which is opened by the **Preferences**, the third set of options is used to customize the default aspect of *2D plots*. The first tab is used to set some general options. Most of them are obvious to understand. If *autoscaling* is set, the scales of the axes will be reset to their default values each time a modification is done on the data series. The *scale font* option is set by default, in this case the size of the font are modified each time the window size is modified.

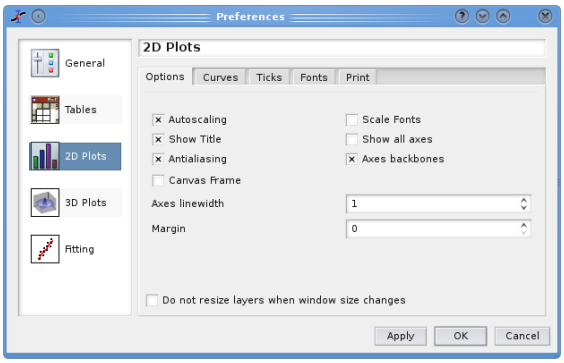
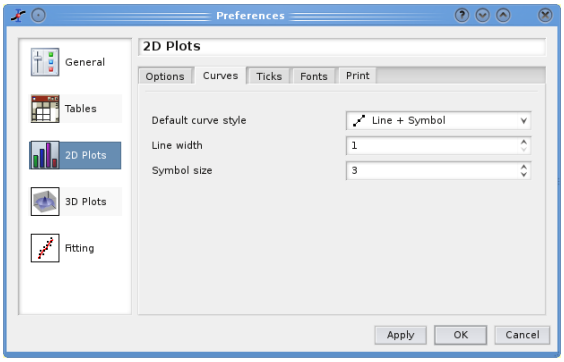
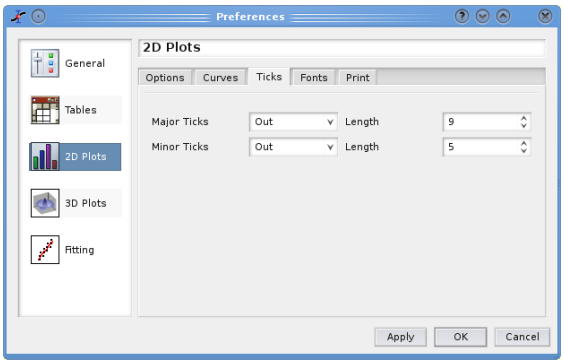


Figure 2.18: The preferences dialog: 2D plot options.

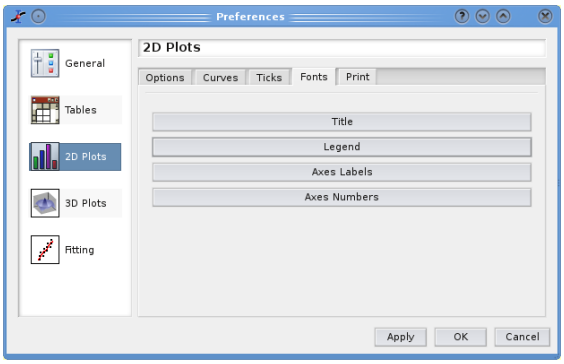
The second tab named *Curves* defines the default style used when you create a new plot.



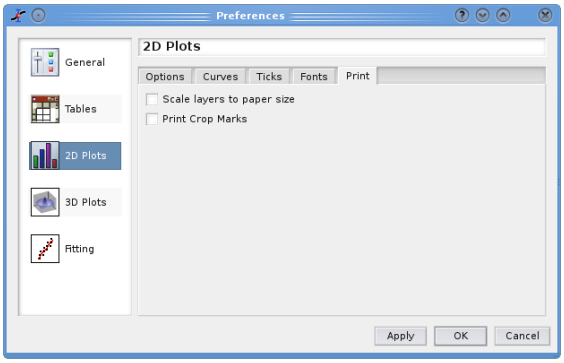
The third tab named *Ticks* defines the default style for the ticks of the axes used when you create a new plot.



The fourth tab named *Fonts* defines the default style for the fonts used for the axes, used when you create a new plot.



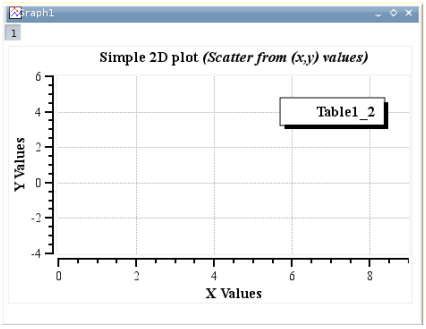
The last tab allows to modify two parameters for the printing of plots. The first one is used to re-scale the plot in order to fit the chosen paper size, the other one to print crops marks around the plot (for cutting).



2.1.6 Working with templates

If you want to build several plots based on the same model, you can use template files. This allows to save geometry of plots, the values, fonts and colors of labels, etc (see Open Template command for details on the items which are saved).

In the following example, the pristine figure is the simple 2D plot presented above, it was saved as a template and an empty plot was created by the **Open Template**.



You just have to add curves with the Add/Remove Curve command, but the style used to draw the curves is not kept in the template.

2.2 Other special 2D plots

2.2.1 Pie plots

A pie plot can be built from two columns in a table, the first column will be considered as text and the second as numbers. By default, each sector of the plot will have one label containing the percentages computed from the Y values. Theses labels can be modified as any other text label.

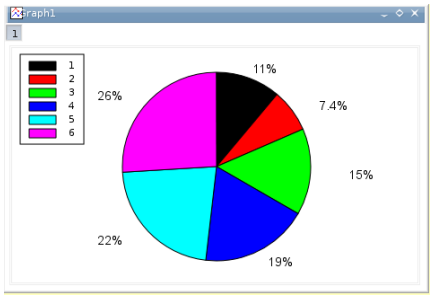


Figure 2.19: An example of pie plot.

2.2.1.1 Formatting of pie plots

These commands are available for the plots generated by the Pie command. The first tab allows the customization of the pie segments. The left fields are used to modify the border which is drawn round each segment: color, type and width of line. The default is no border (line width = 0).

The right fields are used to define the filling of the plots. The color button defines the one used for the first segment, then the others segments will have colors which follow the order defined in the list. The default value for this field is black, so segment 2, 3, etc will be red, green, etc.

The pattern will be used for all segments of the pie, the default value is solid filling. The last field defines the size of the pie in pixels.

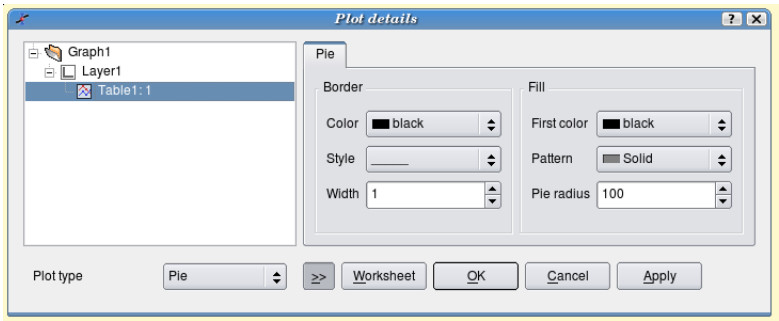


Figure 2.20: Pie segment formatting.

2.2.2 Vectors plots

A vector plot can be built from four columns in a table. The two first columns define the position of each arrow in the X-Y drawing area. The two other columns define the length of the arrows, two methods are available for this:

- *Vector-XYXY:* the two last columns define the position of the arrow while the two first columns define the origin of the arrow.
- *Vector-XYAM:* the two last columns define the angle and the magnitude of the arrow. In this case, the two first columns define the position of the arrow by its origin, its center or its end depending on the options used (see below).

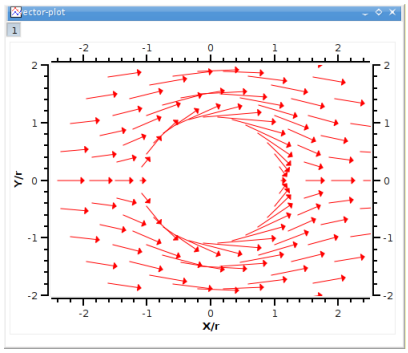


Figure 2.21: An example of a vector plot (fluid flow around a cylinder in a laminar mode).

2.2.2.1 Formatting of vector plots

In the case of a Vector-XYXY plot, the options window allows to modify the shape and size of the arrow head, and also the linestyle used to draw the arrows.

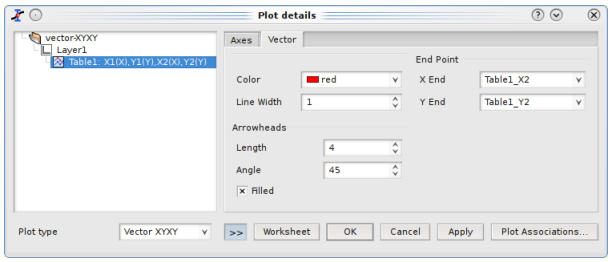


Figure 2.22: Vector-XYXY formatting.

In the case of a Vector-XYAM plot, the options are the same as above. In addition, the relative position of the arrow as a function of the X-Y values can be specified.

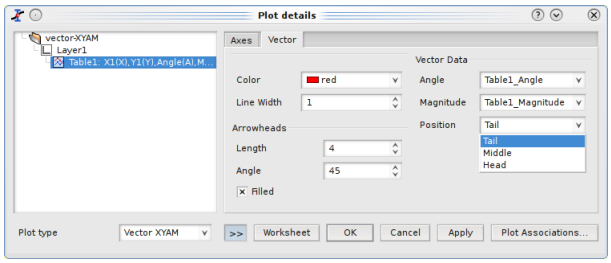


Figure 2.23: Vector-XYAM formatting.

2.3 Statistical plots

Statistical plots are different from conventional 2D-plots since they are not use to show the data themselves. Instead, they are able to present the results of some statistical analysis of the data. Following this, histogram are completely different from the plots obtained by the Vertical Bars command.

2.3.1 Box plots

2.3.1.1 Description of box plots

Box plots are used to show some statistical values which are significant parameters of the distribution of the data. Let's assume that we have a table with 12 values in a column. If you select this column and build a box plot with the Statistical graphs→Box Plot command, you will obtain a graph which is close to the one presented in the figure 2.24. By default, the values which are computed from your data are (figure 2.24):

- Y_{max} The maximum value of Y
- $Y_{5\%}$ The value of Y corresponding to the top 5% of the distribution of numbers
- $Y_{25\%}$ The value of Y corresponding to the top 25% of the distribution of numbers
- $Y_{50\%}$ The value of Y corresponding to the top 50% of the distribution of numbers (also known as the median value)
- Y_{mean} The average value of Y
- $Y_{75\%}$ The value of Y corresponding to the top 75% of the distribution of numbers
- $Y_{95\%}$ The value of Y corresponding to the top 95% of the distribution of numbers
- Y_{min} The minimum value of Y

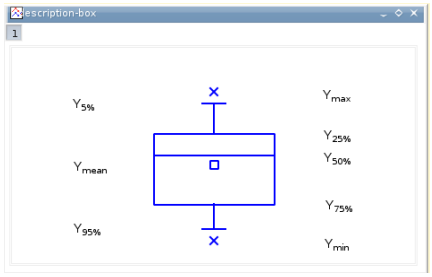


Figure 2.24: An example of a box plot for three columns.

All these parameters give informations on the distribution of data in the column. For example, the difference between Y_{mean} and $Y_{50\%}$ is an indication of the symetry of the distribution. Statistical parameters can be used also to compare distribution of data, you just have to select all the columns and build the box plot.

2.3.1.2 Customization of box plots

There are two ways to modify a box plot: you can modify the statistical parameters which are shown. As in all other plots, you can also modify the appearance of the graphic items.

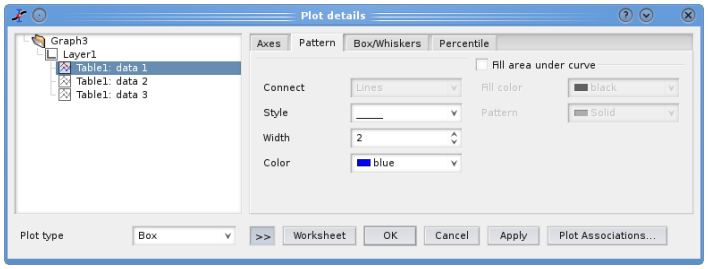


Figure 2.25: The Custom Curves Dialog for box: pattern formatting.

This tab is used to modify the aspect of the box and of the upper and lower whiskers which are attached to it. You can also remove the box and/or the whiskers.

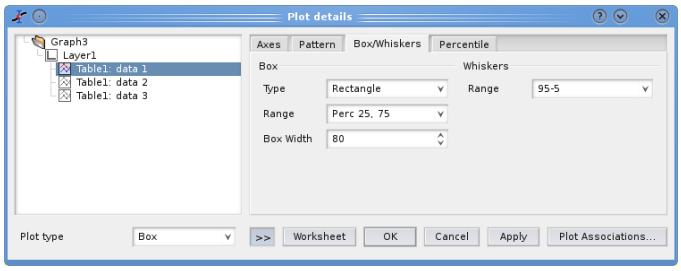


Figure 2.26: The Custom Curves Dialog for box: whiskers formatting.

As explained above, the default is to draw 3 symbols corresponding to Y_{min}, Y_{mean} and Y_{max}. These symbols can be modified (or removed) here. Moreover, you can add two other symbols corresponding to $Y_{99\%}$ and $Y_{1\%}$.

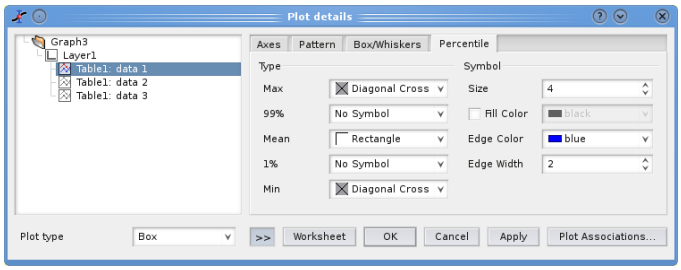


Figure 2.27: The Custom Curves Dialog for box: percentile formatting.

2.3.2 Histograms

2.3.2.1 Building of an histogram

An histogram can be used to show the distribution of the values, that is the numbers of values which are in given intervals. Let's assume that you have a set of data in a column. You can select this column and use the Statistical graphs→Histogram command. After some customization (see next section), you can obtain a plot like the one presented in the figure 2.28.

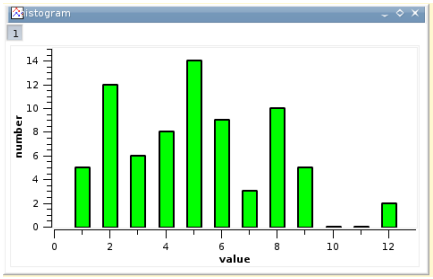


Figure 2.28: An example of histogram.

2.3.2.2 customization of histograms

As for other plots, you can access to the dialog plot through the Plot command of the Format menu. You can also use the other commands of the Format menu to modify axes, labels, titles, etc. The first tab can used to modify the appearance of the columns: lines and filling.

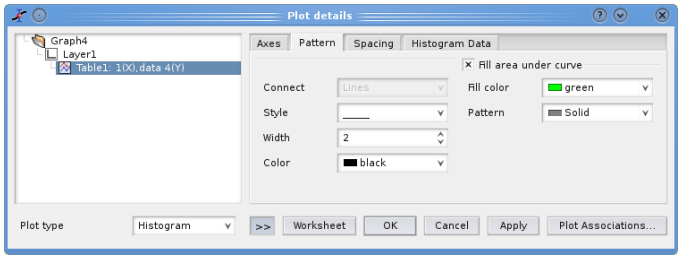


Figure 2.29: Pattern formatting in histograms.

The second tab allows to modify the geometrical parameters of the columns. The parameter *gap between bars* define the distance between two adjascent columns. This is not a true distance, it define the fraction of space which is occupied by the intervalles between columns. By default, this parameter is at 0% so that there is no space between columns. In the example of figure 2.28, a value of 50% has been used, so that the width of space and columns are equal.

the second parameter *Offset* can be used to shift the bars from their default position. For example, In the figure 2.28, the number of values between 3 and 4 is 6, and the corresponding column should be plotted at an abscissae of 3.5. In order to have this column corresponding to the value X=3, a negative shift has been applied. The value of the shift is a percentage of the width of the column, the maximum width of the columns is $\Delta X=1$ in this example and a a gap of 50% is used so a value of -100% has been used, corresponding to a shift $\Delta X=-0.5$.

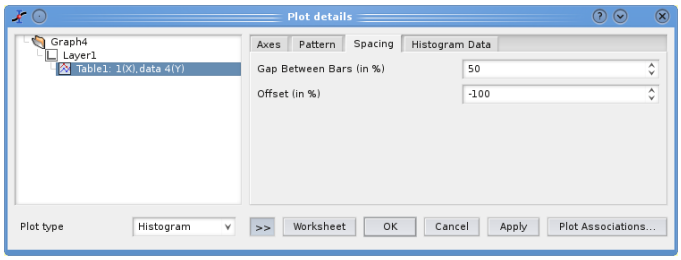


Figure 2.30: Whiskers formatting in histograms.

The last tab is used to define the number of columns used for the plot. It is defined by the X range used for the statistical analysis, and the size of each interval. The default is to use 10 interval in the range $[Y_{min}:Y_{max}]$.

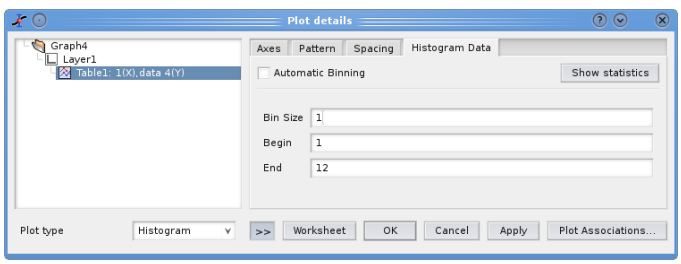


Figure 2.31: Interval selection in histograms.

2.4 3D plots

3D plot are generated from data defined as *Z=f(X,Y)*. As for 2D plots, there are two ways to obtain a 3D plot depending on the way the (X,Y,Z) values are defined:

- You can have your Z values in a matrix. SciDAVis will consider that all the data present in the matrix are Z values, and the X and Y values can be defined as a linear function of the columns and rows numbers.

  The data in the matrix can be entered in several ways:

  - one by one from the keyboard,
  - by reading an ascii file in a table and converting the table into a matrix,
  - by setting the values with a function.

- If you want to plot a function, you don't need a matrix. You can use directly the New→New 3D Surface Plot command.

There are several kinds of 3D plots which can be selected, see the plot3d menu section of the reference chapter for a list of the availables plots.

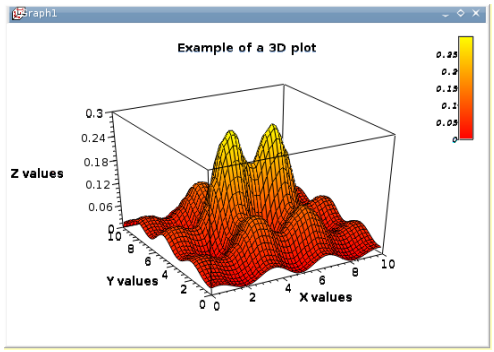


Figure 2.32: Example of a 3D Plots.

The 3D plots use OpenGL so you can easily rotate, scale and shift them with the mouse. Via the 3D plot settings dialog or via the Surface 3D Toolbar you can change all the predefined settings of a three dimensional plot: grids, scales, axes, title, legend and colors for the different elements.

There are several types of plots which can be built from a matrix. They are presented in the plot3d menu

2.4.1 Direct 3D plot from a function

This is the simplest way to obtain a 3d plot. It is done with the New→New 3D Surface Plot command from the File menu or directly with the CTRL+ALT+Z. This will open the following dialog box:

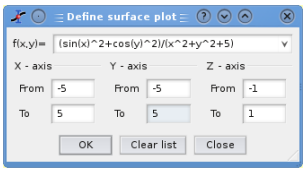


Figure 2.33: Definition of a new surface 3D plot

You can enter the function z=f(x,y) and the ranges for X, Y and Z. Then SciDAVis will create a default 3d plot:

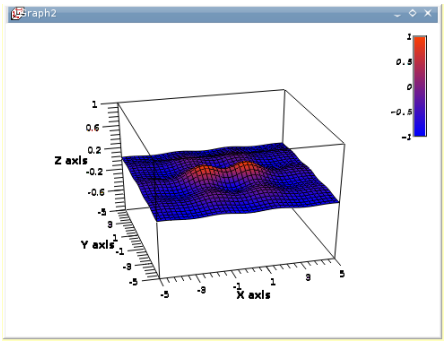


Figure 2.34: The 3D surface plot created by default

You can then customize this plot by opening the Surface plot options dialog. You can modify the axis ranges and parameters, add a title, change the colors of the different items, and modify the aspect ratio of the plot. In addition, you can use the different commands of the 3D surface toolbar to add grids on the walls or to modify the style of the plot. After some modifications, you can obtain the plot presented above.

If you want to modify the function itself, you can use the **surface...** command which can be activated from the context menu with a right click on the 3D plot. This will re-open the *define surface function dialog box*.

The 3D plotting system uses openGL, therefore these plots can be manipulated with the mouse:

- by clicking on the left button and moving the mouse, you can change the viewpoint of the plot. You can come back to the default viewpoint by clicking on the icon of the 3D surface toolbar.
- The ??? can be used to zoom or unzoom the plot. You can come back to the default zoom value by clicking on the icon of the 3D surface toolbar.

2.4.2 3D plot from a matrix

The second way to obtain a 3D plot is to use a matrix. Therefore, the first step is to fill the matrix. This can be done by defining a function.

The New→New Matrix command create a default empty matrix with 32x32 cells. Then use the Dimensions command from the Matrix menu to modify the number of rows and columns of the matrix. The Set Coordinates command can then be used to define the X and Y ranges.

Then use the Assign Formula command to fill the cells with numbers. The ranges of X and Y defined in the previous step are not known by this dialog box, then the function is defined with the row and column numbers (i and j) as entry parameters.

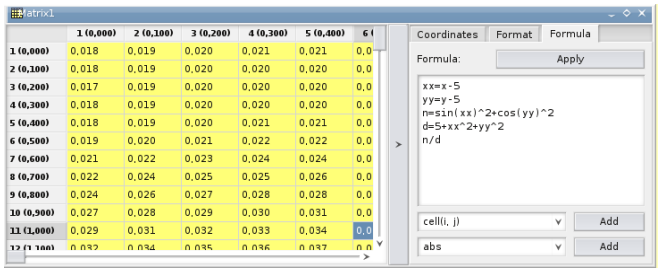


Figure 2.35: Assigning a multi-lines formula to a matrix

The other way to obtain a matrix is to import an ASCII file into a table with the Import Ascii command from the File menu. The table can then be transformed in a matrix with the command Convert to Matrix command from the Table menu.

You can then use this matrix to build a 3D plot with one of the command of the Plot menu.

2.4.3 Customization of a 3D plot

A dialog with five tab is activated by double clicking on a contour curve (or on the plotting area) of a 3D plot. It can also be accessed by the commands of the Format menu.

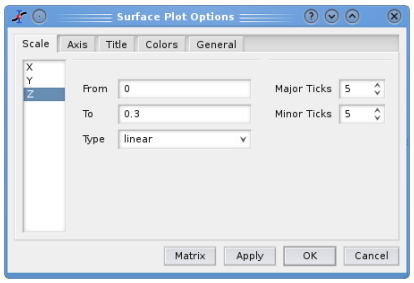
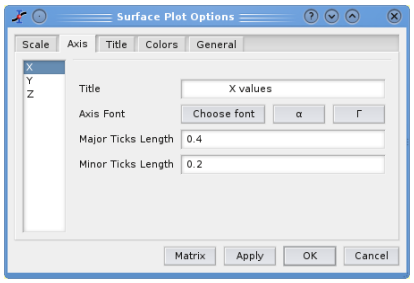


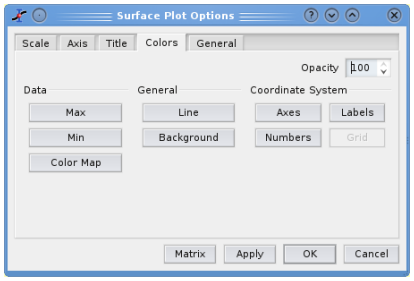
Figure 2.36: The Contour curves options dialog.

The first group of settings *Scales* is used to define the scales of the three axis. It works in the same way as scaling of 2D plots.

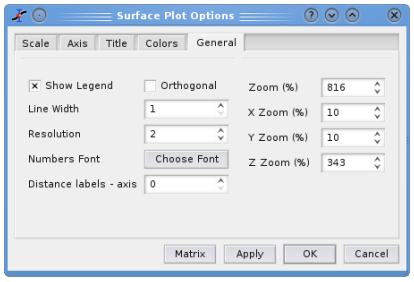


The second tab *Axis* is used to define the labels of the three axis. You can also customize the size of the ticks, beware that this size is given in real units. Therefore, it should be chosen in relation to the axis ranges: for example, if you put a length of 1 for the ticks of the X-axis, the length will correspond to a unit of 1 from the Y axis. In the same way, Y and Z ticks are computed in reference to X range.

The third tab *Title* is used to modify the title of the plot. Compared to conventional label dialog box of SciDAVis, it exhibits some limitations related to the 3D drawing system (no subscripts, superscript, no bold or italic characters). See the Title command for more details.



The fourth tab *Colors* is used to modify the color of the different elements. For *General* and *Coordinate System* elements, it is a conventional choosing color dialog box. You can also customize the colormap used to draw the data. See the next section for more details on color maps.



The last tab *General* is used to modify some global parameters of the plots. The *orthogonal* check box allows to change the 3D view from conventional perspective to orthogonal view. It correspond to the icon of the 3D surface toolbar. The parameter *resolution* is 1 by default, it indicates that all data points are used to draw the contour

curves. If the line network is too dense, you can increase this parameter: with a value of 2 only 1 value over 2 will be used.

By default, the plot use the same graphic scales for the three axes. If the ranges are very different, you can adjust the size of the plot by changing the zoom over the different axes. In the example presented above, X and Y ranges are 10 while Z range is 0.2, then a zoom of 3000% should be used for Z axis if the zooms on X and Y are kept at 100%. You can also use the icon of the 3D surface toolbar to adjust automatically these zoom values.

2.4.3.1 Modification of color schemes

The two colors (data min and data max) defines the color scheme which is used to show the Z-values. They are the colors used for the minimum value of Z (Z_{min}) and the maximum value of Z (Z_{max}). We can define the colors by their Red, Green and Blue parameters: [R,G,B]. Then, a value Z will be represented by a color defined as a linear interpolation:

$$R = R_{min} + \frac{(R_{max} - R_{min})}{(Z_{max} - Z_{min})}(Z - Z_{min})$$
$$G = G_{min} + \frac{(G_{max} - G_{min})}{(Z_{max} - Z_{min})}(Z - Z_{min})$$
$$B = B_{min} + \frac{(B_{max} - B_{min})}{(Z_{max} - Z_{min})}(Z - Z_{min})$$

The default colors for Z_{min} and Z_{max} are respectively blue ([R,G,B] = [0,0,255]) and red ([R,G,B] = [255,0,0]). This lead to the following color scheme:

Another classical color scheme can be built with Z_{min} = [160,32,32] and Z_{max} = [255,255,0] (yellow). It leads to:

Another way to define colors is to read a colormap from a file. The format of the file is simple: each line defines a color by red, green and blue values as integers between 0 and 255. The numbers should be separated by spaces. You can find several examples of colormaps on the QwtPlot3D web site.

2.4.4 Changing default 3D plot options

Most of the parameters presented in the previous section can be set by default with the Preferences command of the Edit menu.

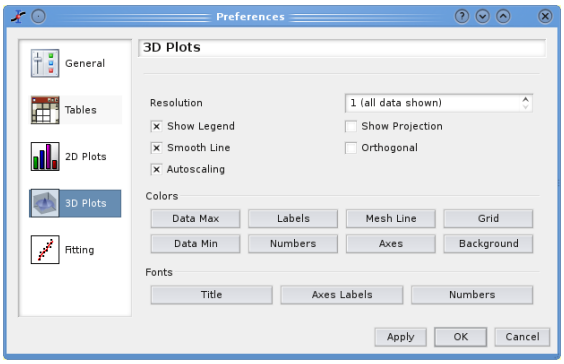


Figure 2.37: The preferences dialog: 3D plot options.

2.5 Multilayer Plots

The multilayer windows can contain multiple plots (layers) with different characteristics. Each layer has a corresponding button, which displays a number and is pressed when the layer is the currently active layer. There is only one active layer at a time, and the plot tools (zoom, cursors, drawing tools, delete and move points) can only operate on this layer. Each plot can be made active by clicking on it or on its corresponding button.

To arrange the layers use the Arrange Layers command. You can add or remove layers with the Add Layer command and Remove Layer command or copy/paste layers from one multilayer window to another. All these functions can be reached via the Graph menu, by using the Plot toolbar or via the context menu (right click in the multilayer window anywhere outside a plot area).

You can resize and move a layer using the Layer geometry dialog. You can also arrange and resize the plots by hand. A whole plot can be moved by drag and drop: click on the plot and keep the left mouse button pressed.

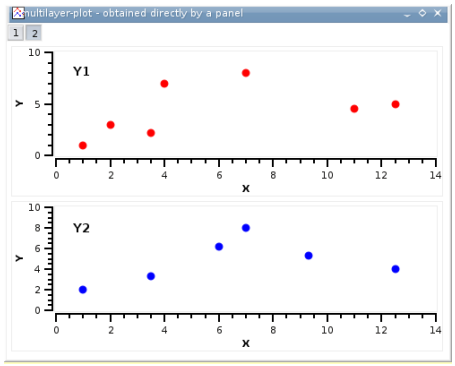
By keeping the Shift key pressed and dragging the border of a plot you can scale a plot as needed. When moving the mouse over the borders of a plot, you will see the corresponding arrows.

You can also use the mouse wheel in order to resize the layers: keeping the Ctrl key pressed and scrolling will resize the hight of the plot canvas, while keeping the Alt key pressed and scrolling will resize its width. By keeping the Shift key pressed and scrolling you can resize the plot in both dimensions.

2.5.1 Building a multilayer plot panel

This is the simplest way to obtain a multilayer plot. It can be used if you want to build a panel of plots with a simple arrangement: 2 plot in a row or in a column, or 4 plots in 2 rows and 2 columns.

You can select two columns with Y-values in a table, and then use one of the **Panel** commands in the Plot menu. SciDAVis will create a panel of plots in which the size of the different elements of each plot are synchronized.

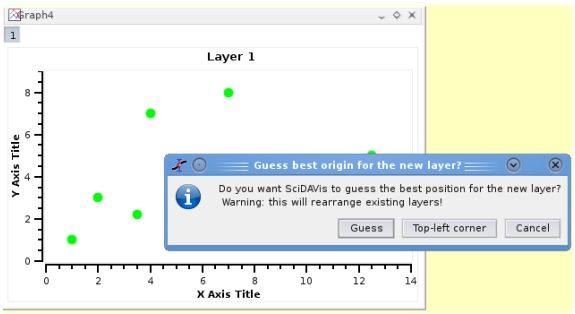


You can then customize the two plots, if you want to change the arrangement of the panel, you can use the Arrange Layers command from the Graph menu. It must be reminded in this case that each plot is in a layer with a surface which is the half or the quarter of the window surface area. So, if you want to share an element between the two plots (for example a text label), you need to add it in a new layer (see the Add Text command for more detaile).

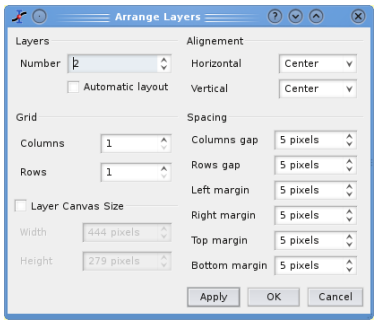
2.5.2 Building a multilayer plot step by step

If you need to build a more complex multilayer plot, you can define it step by step.

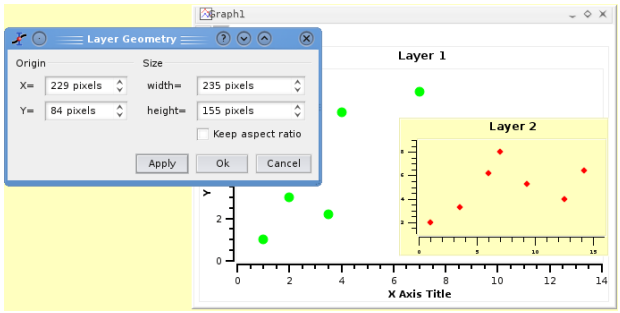
The first step is to build your first plot, for example from two columns of a table. We obtain a standard plot window:



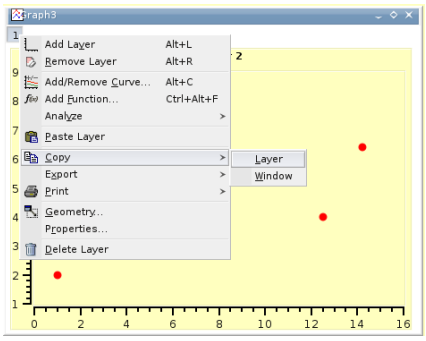
Then, select the plot window and use the Add Layer command from the Graph menu. This will activate a dialog box. If you choose "Guess" you will obtain a panel with two columns, if you choose "corner" you will obtain two superposed layers. If you want to build a panel with two rows or some other regular matrix of plots, you can use the Arrange Layers command to convert the plot to a panel.



If you want to build a more complex geometry like plots inserted in another one, you can modify the geometry of each plot with the *layer geometry* command. This command is accessible with the context menu of the layer and it allows to modify the size and the position of each layer. Beware that by default, the background of the different layers are not transparent. Therefore, you must modify this parameter if you want to have some superposition of plots. This modification can be done with the Plot command of the Format menu.



For each layer, you can use the Add/Remove Curve command to select the X and Y values from one of the tables of the project.



After this, you can customize your plot. At the end, the modifications done on the axis or on the axis labels may have modified the geometry of the two plots. You can synchronize again the two plots by applying again the Arrange Layers command.

2.6 Adding objects to a plot

2.6.1 Adding a text label

This dialog can be opened by several commands such as Title command or when you double click on a text object in your plot. It allows to add/customize the text objects.

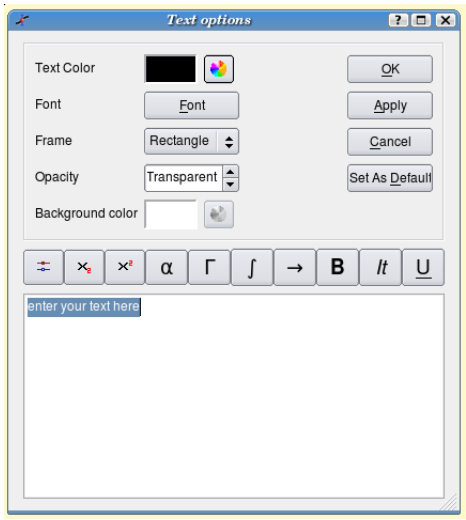
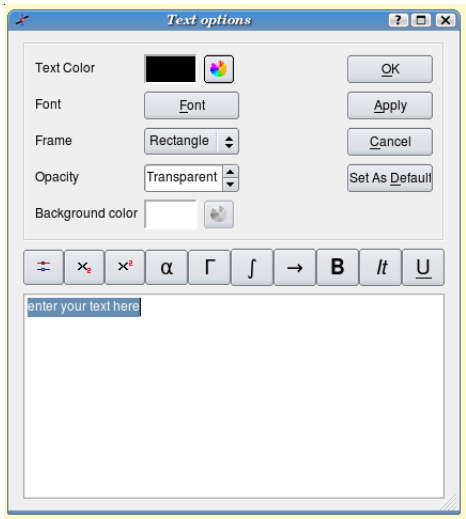


Figure 2.38: The text options dialog.

The *Color*, *Font* and *Alignment* commands allow the modification of the general settings of the text label.

The text item can be modified in the text window. Several improvements can be added to the text:

- _{text} will draw the text as subscripts. You can insert this sequence by clicking on the .
- ^{text} will draw the text as superscripts. You can insert this sequence by clicking on the .
- By clicking on the , you can open a new dialog which allows to select greek characters:



- By clicking on the Γ, you can open a new dialog which allows to select various mathematical symbols:

- text will draw the text with bold characters. You can insert this sequence by clicking on the B.
- text will draw the text with italic characters. You can insert this sequence by clicking on the It.
- <u>text</u> will draw the text with underlined characters. You can insert this sequence by clicking on the U.

Chapter 3

Analysis of data and curves

3.1 Fast Fourier Transform

This function can be accessed by the FFT command of the Analysis-tables menu when a table is selected, or Analysis-plots menu when a plot is selected. The Fourier transform decomposes a signal in its elementary components by assuming that the signal x(t) can be describe as a sum:

$$x(t) = \sum_n a_n \cos(\omega_n t + \psi_n)$$

EQUATION 3.1.1: Fourier equation

in which ω_n are the frequencies, a_n are the amplitudes of each frequency and ψ_n are the phase corresponding frequency. SciDAVis will compute these parameters and build a new plot of the amplitude as a function of the frequency. FFT can be performed on a curve to extract the characteristic frequencies.

Let's assume you have the signal presented in the next figure. You can select the FFT command of the Analysis-plots menu to open the FFT dialog box.

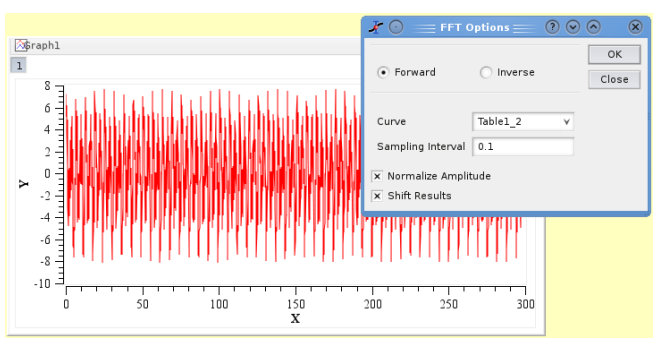


Figure 3.1: A signal and the FFT dialog box for a plot.

If the *Normalize Amplitude* check box is on, the amplitude curve is normalized to 1. If the *Shift Results* check box is on, the frequencies are shifted in order to obtain a centered x-scale. By default, the *Sampling Interval* corresponds to the interval between X-values. Giving a smaller value makes no sense, but you can increase this value in order to sample less values.

SciDAVis will create a new plot window with the FFT amplitude curve, and a new table which contains the real part, the imaginary part, the amplitude, and the angle of the FFT. In this example, the amplitude curve has been normalized, and the frequencies have been shifted to obtain a centered x-scale.

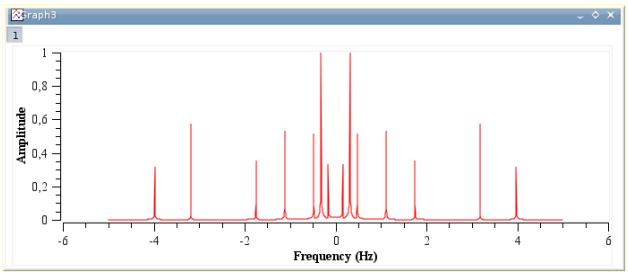


Figure 3.2: The resulting FFT with the characteristic frequencies.

In the case of a table, you must select the sampling column (X-values) and one columns (for real numbers) or two columns (for complex numbers) for Y-values.

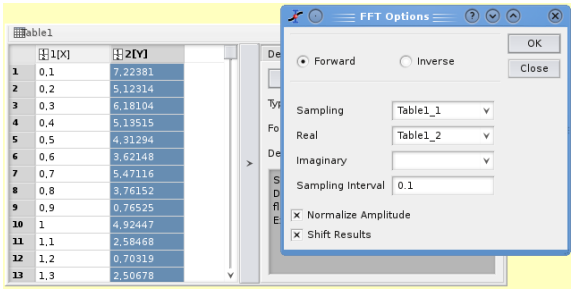


Figure 3.3: The **FFT** dialog box for a table.

3.2 Filtering of data curves

In this section, it will be assumed that you have the signal presented in the previous section (see figure Figure 3.1). We can analyze this signal by doing a FFT on the data curve and it will show that this signal has a power spectrum with high and low frequencies (see figure Figure 3.2). The newt sections will show the influence of the different filters on this data curve.

3.2.1 FFT low pass filter

This filter allows to cut the high frequencies of a signal. You just have to select the cut-off frequency of the filter. Let us assume that we want to keep the frequencies below 1.5 Hz, we will obtain:

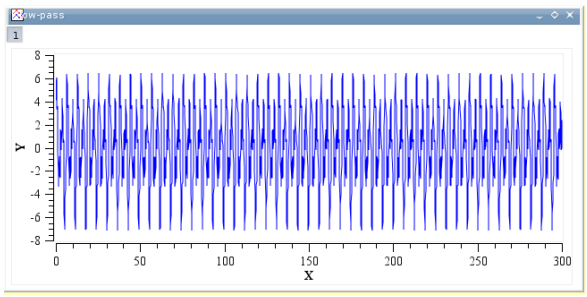
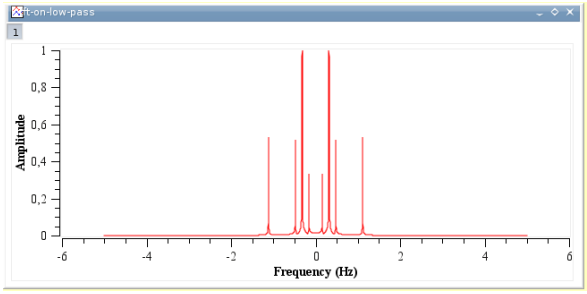


Figure 3.4: Signal after a FFT low pass filter

The power spectrum of this new signal shows that the frequencies below 1.5 Hz have been kept.



3.2.2 FFT high pass filter

This filter allows to cut the low frequencies of a signal. You just have to select the cut-off frequency of the filter. Let us assume that we want to keep the frequencies above 1.5 Hz, we will obtain:

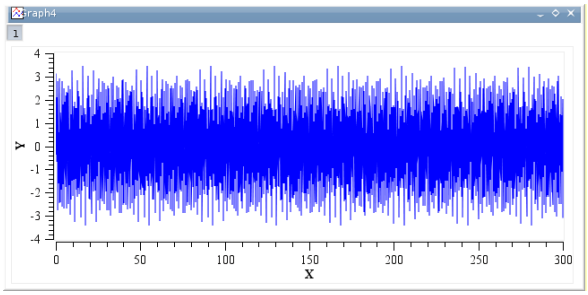
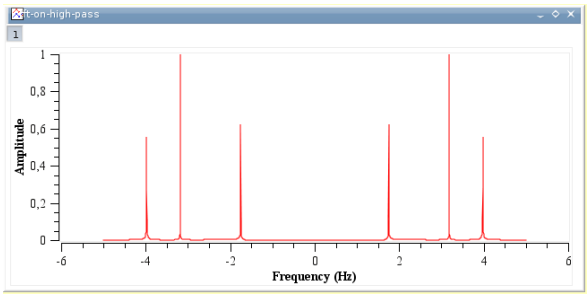


Figure 3.5: Signal after a FFT high pass filter

The power spectrum of this new signal shows that the frequencies above 1 Hz have been kept.



3.2.3 FFT band pass filter

This filter allows to cut the low and high frequencies of a signal. You just have to select the high and low cut-off frequencies of the filter. Let us assume that we want to keep the frequencies between 1.5 and 3.5 Hz, we will obtain:

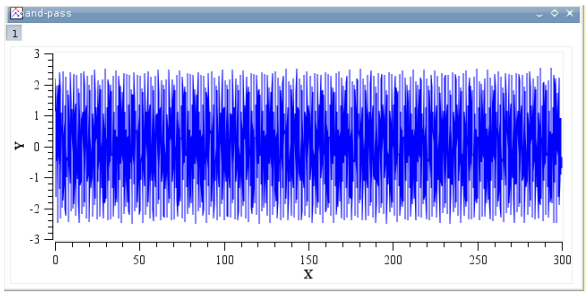
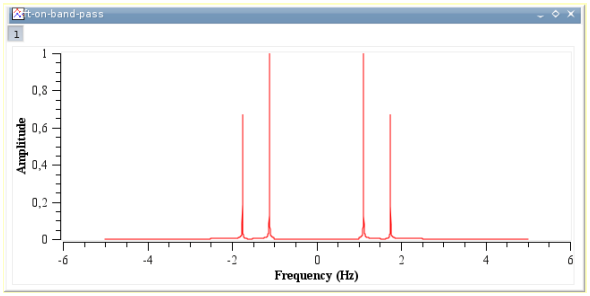


Figure 3.6: Signal after a FFT band pass filter

The power spectrum of this new signal shows that only the frequencies at 1.5 and 3.5 Hz have been kept.



3.2.4 FFT block band filter

This filter allows to keep the low and high frequencies of a signal. You just have to select the high and low cut-off frequencies of the filter. Let us assume that we want to remove the frequencies between 1.5 and 3.5 Hz, we will obtain:

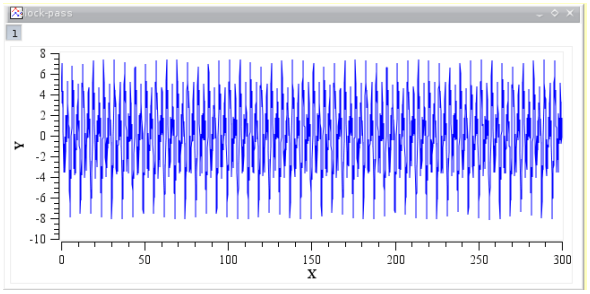
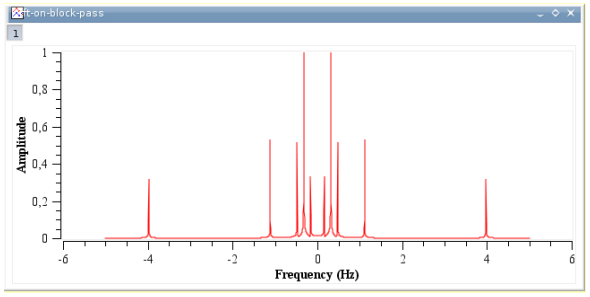


Figure 3.7: Signal after a FFT block band filter

The power spectrum of this new signal shows that only the frequencies below 1.5 Hz and above 3.5 Hz have been kept.



3.3 Correlation and autocorrelation

This function can be accessed by the Correlate command of the Analysis-tables menu when a table is selected. The correlation function, also known as the covariance func-

tion is used to test the similarity of two signals *x(t)* and *y(t)*. It is computed by:

$$R(\tau)=\overline{(x(t)-\bar{x}).(y(t+\tau)-\bar{y})}$$

EQUATION 3.3.1:

in which $\bar{x}$ and $\bar{y}$ are the mean values of the signals *x(t)* and *y(t)* respectively. If the number of points is *N*, the function will be computed between *-N/2* and *N/2*. The abscissas are therefore point numbers and not *t* values.

To perform a cross correlation between two signal, they must be in the same table and use the same abscissa. You just have to select the two columns in the table, and select the Correlate command from the Analysis-tables menu. A plot will be created and the values of the correlation function will be added as two new columns in the table.

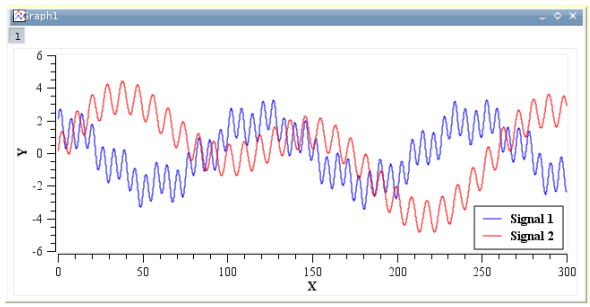


Figure 3.8: An example of a correlation between two functions: the two signals.

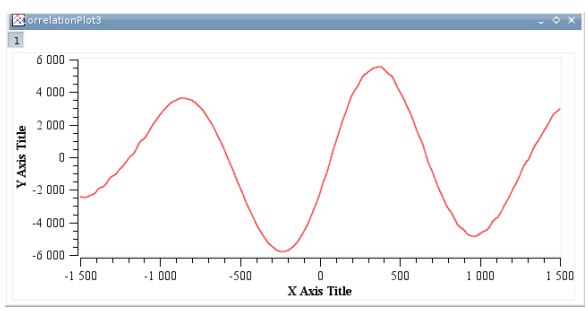


Figure 3.9: An example of a correlation between two functions: the correlation function.

The correlation of a signal with itself can also be used in spectral analysis (it is then called autocorrelation or autocovariance function). This operation can be performed by selecting one column in a table and use the Autocorrelate command from the Analysis-tables menu.

3.4 Convolution of functions

This function can be accessed by the Convolute command of the Analysis-tables menu when a table is selected. The convolution of two functions $f_1(x)$ and $f_2(x)$ is the function defined by:

$f_1(x)$ is the signal and $f_2(x)$ is the transfer function.

3.5 Deconvolution

This function can be accessed by the Deconvolute command of the Analysis-tables menu when a table is selected. The deconvolution is the inverse of convolution, that is finding the function $f_1(x)$ which is the solution of the equation $f_1 * f_2 = g$.

3.6 Fitting of data and curves

Fitting can be done in two ways:

- A general *Fit Wizard* which allows to use complex functions and to adjust the fitting parameters.
- A set of simplified fitting dialog boxes for most used functions like exponential growth or decay, etc.

3.6.1 Non Linear Curve Fit

This function can be accessed by the Fit Wizard command of the Analysis-plots menu when a plot is selected, or the Analysis-tables menu when aa table window is selected. In the latter case, this command first creates a new plot window using the list of selected columns in the table.

This Command is used to fit discrete data points with a mathematical function. The fitting is done by minimizing the least square difference between the data points and the Y values of the function.

Note:
If the data points are modified, the fit is not re-calculated. Then, you need to remove the old fitted curve and to redo the fit with the same function and the new points.

The top of the dialog box is used to choose a function among the one which are already define. Four types of functions are available: the user defined functions which have been saved, the classical functions proposed by SciDAVis in the analysis menu, the simple elementary built-in functions, and external functions via pluggins.

To choose one of these functions, you just have to select it and to click on the checkbox under the selector.

If you want to define your own function, you can use the bottom half of the dialog box. You can write you own mathematical expression or add expressions obtained with

the function selector. Then you need to define the parameters which have to be fitted in a comma separated list.

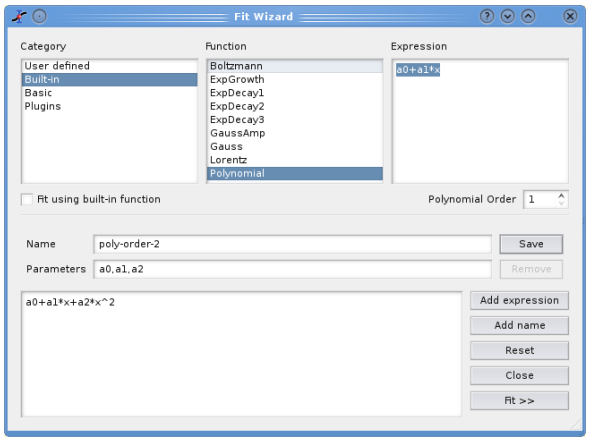


Figure 3.10: The first step of the **Fit Wizard** dialog box.

The second step is to define the parameters for the fit. You have to give initial guess for the fitting parameters.

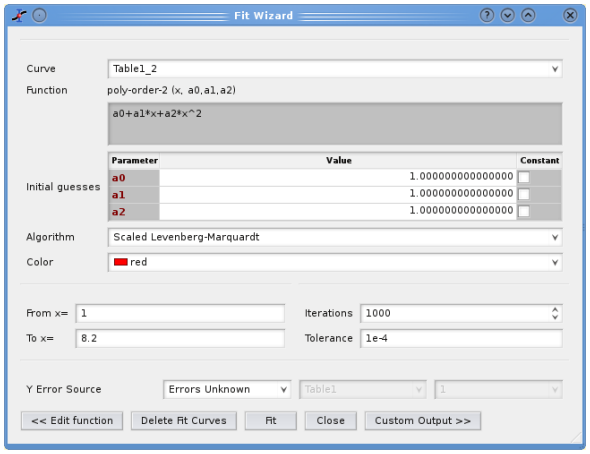


Figure 3.11: The second step of the **Fit Wizard** dialog box.

In this second tab you can also choose a weighting method for your fit (the default is *No weighting*). The available weighting methods are:

1. *Instrumental*: the values of the associated error bars are used as weighting coefficients. You must add Y-error bars to the analyzed curve before performing the

fit.

2. *Statistical*: the weighting coefficients are calculated as the square-roots of each data point in the fitted curve.

3. *Arbitrary Dataset*: you have the possibility to set the weighting coefficients using an arbitrary data set. The column used for the weighting must have a number of rows equal to the number of points in the fitted curve.

After the fit, the log window is opened to show the results of the fitting process.

Depending on the settings in the *Custom Output* tab, a function curve (option *Uniform X Function*) or a new table (if you choose the option *Same X as Fitting Data*) will be created for each fit. The new table includes all the X and Y values used to compute and to plot the fitted function and is hidden by default, but it can be found and viewed with the project explorer.

The results are shown in the log window, the curve is plotted in the active window, and a table is created to store the fit.

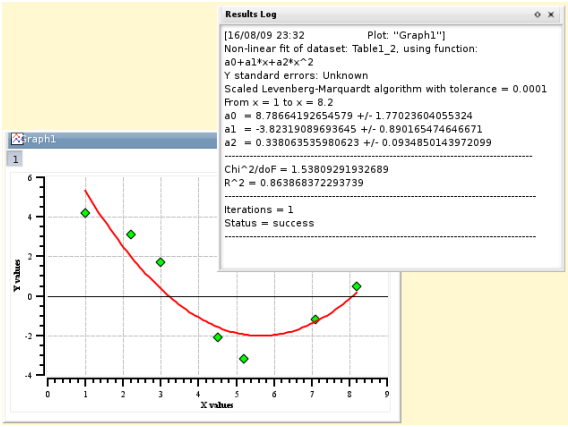


Figure 3.12: The results of the **Fit Wizard**.

3.6.2 Fitting to specific curves

SciDAVis include quick access to the most usefull functions for fitting. Beware that when you use these commands, SciDAVis uses default values as initial guesses for the parameters. Therefore, the convergence may be difficult or even impossible if these initial values are too far from the final values. In this case, you can use the Fit Wizard command or the Fit Wizard command, select the function in the *built-in* set and give good initial values for parameters.

3.6.2.1 Fitting to a line

This command is used to fit a curve which has a linear shape. The results will be given in the Log panel.

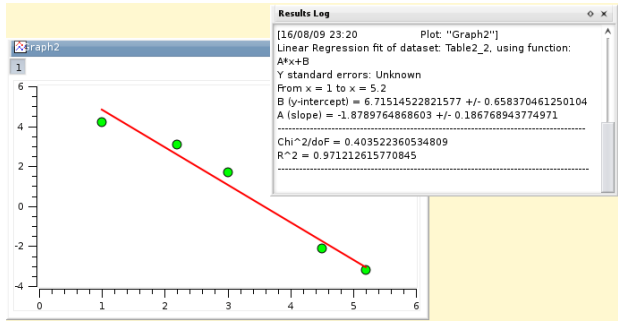
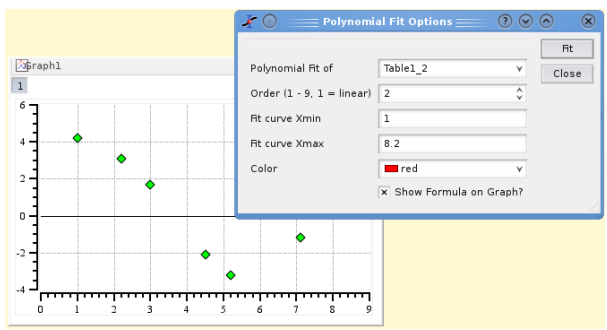


Figure 3.13: The results of a **Quick Fit→Fit Linear**.

3.6.2.2 Fitting to a polynomial

This command is used to fit a curve which has a linear shape. The results will be given in the Log panel



3.6.2.3 Fitting to a Bolzmann function

This command is used to fit a curve which has a sigmoidal shape. The function used is:

$$y=(A_2-A_1)\left(1+\exp\left(\frac{x-x_0}{dx}\right)\right)$$

EQUATION 3.6.1: Bolzmann equation

in which A_2 is the high Y limit, A_1 is the low Y limit, x_0 is the inflexion point and dx is the width.

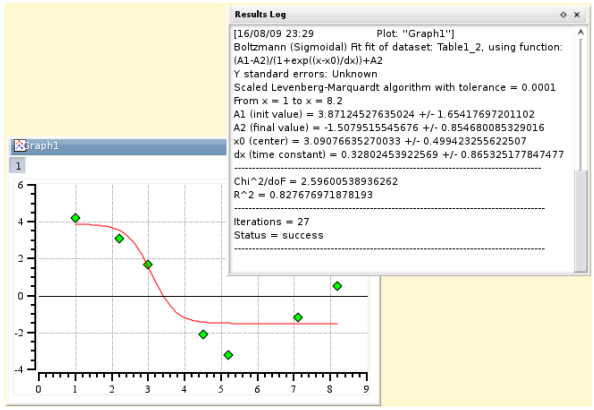


Figure 3.14: The results of a **Quick Fit→Fit Bolzmann (Sigmoïdal)**.

3.6.2.4 Fitting to a Gauss function

This command is used to fit a curve which has a bell shape. The function used is:

$$y = y_0 + A \exp\left(\frac{-(x - x_c)^2}{2w^2}\right)$$

EQUATION 3.6.2: Gauss equation

in which A is the height, w is the width, x_c is the center and y_0 is the Y-values offset.

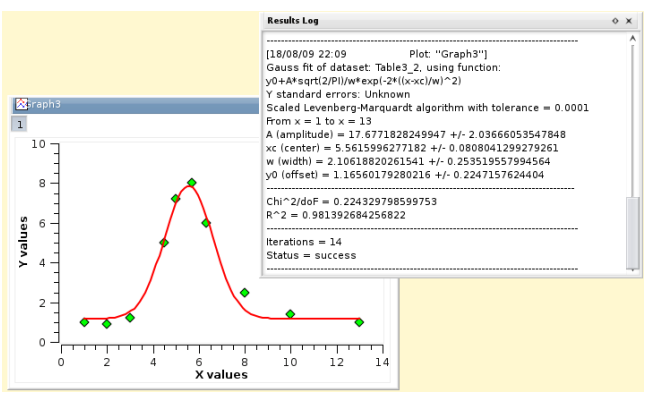


Figure 3.15: The results of a **Quick Fit→Fit Gaussian**.

3.6.2.5 Fitting to a Lorentz function

This command is used to fit a curve which has a bell shape. The function used is:

$$y = y_0 + 2\frac{A}{\pi}\frac{w}{4(x - x_c)^2 + w^2}$$

EQUATION 3.6.3: Lorentz equation

in which A is the area, w is the width, x_c is the center and y_0 is the Y-values offset.

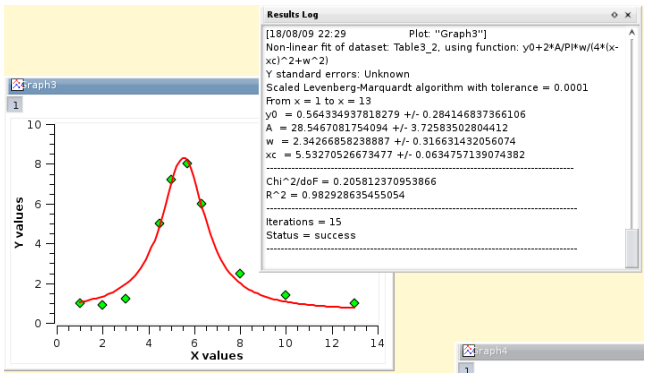


Figure 3.16: The results of a **Quick Fit→Fit Lorentzian**.

3.6.3 Multi-Peaks fitting

This kind of fitting allows to fit your data points to a sum of N Gaussian or Lorentzian functions. The first step is to specify the number of peaks. Then you must define the position of each peak on the curve. This is done by selecting one data point on the plot, then validate your choice for each peak with the *ENTER* key.

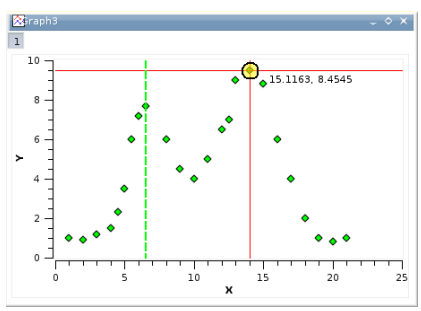


Figure 3.17: The selection of the position of the peaks.

Then, the fitting is done in the same way as for the other quick-fit commands. For the position of the data points used for the selection of the position of the peaks are just initial guesses for the fitting.

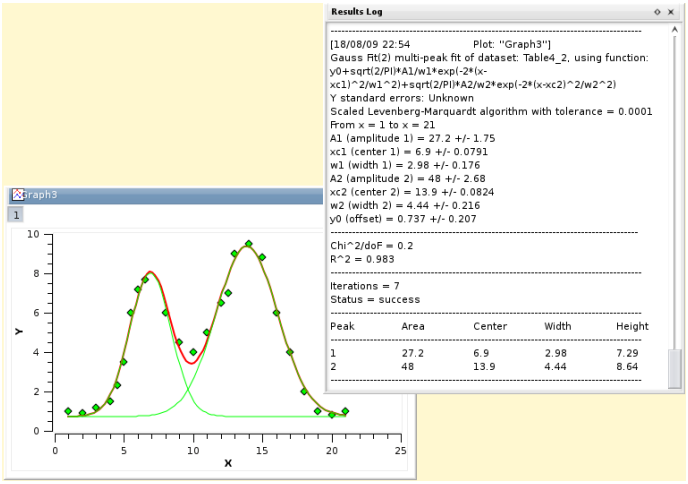


Figure 3.18: The results of a **Quick Fit→Fit Multipeak→Gaussian**.

As for the other quick-fit commands, if you want to fit with a sum of more complex curves (e.g. a combination of lorentzian and gaussian functions), use the Fit Wizard.

3.6.4 Changing default parameters for fitting

This dialog can be accessed by the Preferences command of the Edit menu. It allows to modify the way the fitted curves are drawn on the plots and some options for the presentation of the fitted values. If you want to modify some parameters related to the fitting itself, like the tolerance, you have to do it in the Fit Wizard.

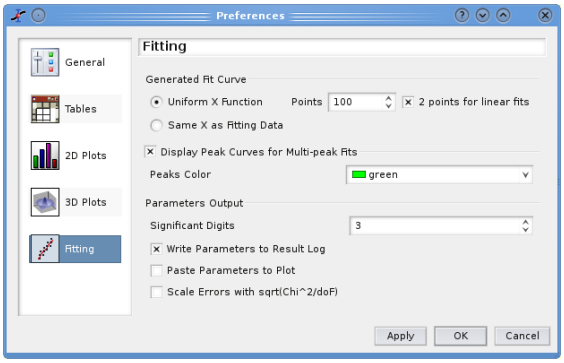
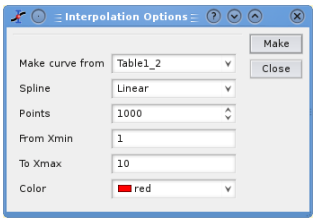


Figure 3.19: The preference dialog for fitting.

3.7 Interpolation

The interpolation command will create a new data curve with a high number of points by interpolation of your data. The dialog box allows to define this number of points (default value = 1000). Then the method used for interpolation, the interval of X-values and the color of the interpolated curve can be chosen. In addition to the new curve in the active plot, a new table will be created.



The simplest interpolation method is the *linear* method. In this case, a linear variation is used to compute the data points between two values. The *cubic* method will use the Cubic Splines method (in this case at least 4 points are needed). The last method *Akima* is a polynomial interpolation. You can refer to the corresponding section of the GNU Scientific Library for more details.

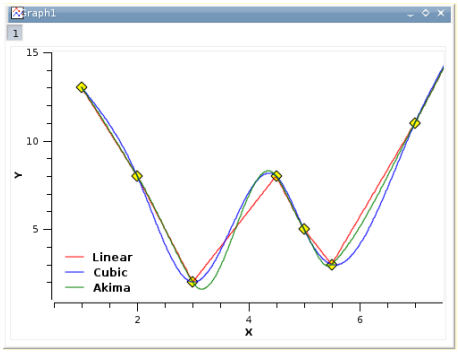


Figure 3.20: Comparison of the three methods of interpolation

Chapter 4

Scripting

SciDAVis supports different interpreters for evaluating mathematical expressions and for executing scripts.

4.1 muParser

The constants _e=e=E and _pi=pi=PI=Pi are defined, as well as the operators and functions listed below. muParser is purely a parser for mathematical expressions and as such does not support executing complex scripts. However, you can make formulas easier to read and modify by assigning sub-expressions to variables and inserting comments.

Variable assignments evaluate to the assigned value, so they can be used in the middle of a formula. They can also stand on a line of their own, or separated from other parts of the formula by a semicolon (;). The result of evaluating the last line of a multi-line formula is taken as the result of the whole formula.

Comments start with a hash mark (#) and run till the end of the line.

It is important to note that the muParser commands listed in this section are intended to be used to work inside the *Formula* tab of a *Table*, i.e. most of them will not work as a *script* in a *Note*. Of course, if one try to execute something like 'a=cos(pi*pi)' in a *Note*, it will run without errors, but the 'a' value will not be known by the user.

4.2 Python

This module provides bindings to the Python programming language. Basic usage in the context of SciDAVis will be discussed below, but for more in-depth information on the language itself, please refer to its excellent documentation.

4.2.1 Getting Started

Make sure your current project uses Python as its interpreter by selecting the menu point Scripting->Scripting Language and double-clicking on "Python" in the resulting dialog (if the dialog appears, but does not contain the "Python" item, your installation

Name	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation (raise a to the power of b)
and	logical and (returns 0 or 1) - works only if SciDAVis is built using muParser 1.x versions
&&	logical and (returns 0 or 1) - works only if SciDAVis is built using muParser >= 2.0.0
or	logical or (returns 0 or 1) - works only if SciDAVis is built using muParser 1.x versions
||	logical or (returns 0 or 1) - works only if SciDAVis is built using muParser >= 2.0.0
xor	logical exclusive or (returns 0 or 1) - works only if SciDAVis is built using muParser 1.x versions; there are no equivalent operator for muParser >= 2.0.0
<	less then (returns 0 or 1)
<=	less then or equal (returns 0 or 1)
==	equal (returns 0 or 1)
>=	greater then or equal (returns 0 or 1)
>	greater then (returns 0 or 1)
!=	not equal (returns 0 or 1)

Table 4.1: Supported Mathematical Operators

Name	Description
abs(x)	absolute value of x
acos(x)	inverse cosine
acosh(x)	inverse hyperbolic cosine
asin(x)	inverse sine
asinh(x)	inverse hyperbolic sine
atan(x)	inverse tangent
atanh(x)	inverse hyperbolic tangent
avg(x1,x2,x3,...)	average value, this command accept a list of arguments separated by commas
bessel_j0(x)	Regular cylindrical Bessel function of zeroth order, $J_0(x)$.
bessel_j1(x)	Regular cylindrical Bessel function of first order, $J_1(x)$.
bessel_jn(n,x)	Regular cylindrical Bessel function of nth order, $J_n(x)$.
bessel_jn_zero(n, s)	n^{th} zero of regular cylindrical Bessel function of nth order, J_n(bessel_jn_zero(n,s))=0
bessel_y0(x)	Irregular cylindrical Bessel function of zeroth order, $Y_0(x)$ for x>0.
bessel_y1(x)	Irregular cylindrical Bessel function of first order, $Y_1(x)$ for x>0.
bessel_yn(n,x)	Irregular cylindrical Bessel function of nth order, $Y_n(x)$ for x>0.
beta (a,b)	Computes the Beta Function, B(a,b) = Gamma(a)*Gamma(b)/Gamma(a+b) for a > 0 and b > 0.
ceil(x)	ceiling; smallest integer greater or equal to x
cos(x)	cosine of x
cosh(x)	hyperbolic cosine of x
erf(x)	error function of x
erfc(x)	Complementary error function erfc(x) = 1 - erf(x).
erfz(x)	The Gaussian probability density function Z(x).
erfq(x)	The upper tail of the Gaussian probability function Q(x).
exp(x)	Exponential function: e raised to the power of x.
floor(x)	floor; largest integer less than or equal to x
gamma(x)	Computes the Gamma function, subject to x not being a negative integer
gammaln(x)	Computes the logarithm of the Gamma function, subject to x not a being negative integer. For x<0, log(|Gamma(x)|) is returned.
hazard(x)	Computes the hazard function for the normal distribution h(x) = erfz(x)/erfq(x).
ln(x)	natural logarithm of x
log(x)	decimal logarithm of x
log2(x)	base 2 logarithm of x
w0(x)	Principal branch of Lambert's W function, $W_0(x)$. W_0 is defined as a solution to the equation $W_0(x)*\exp(W_0(x))=x$. For x<0, there are tow real-valued branches; this function computes the one where W>-1 for x<0 (compare w1(x)).
w1(x)	Secondary branch of Lambert's W function, $W_{-1}(x)$. W_{-1} is defined as a solution to the equation $W_{-1}(x)*\exp(W_{-1}(x))=x$. For x<0, there are tow real-valued branches; this function computes the one where W<-1 for x<0 (compare w0(x)).
min(x1,x2,x3,...)	Minimum of the list of arguments
max(x1,x2,x3,...)	Maximum of the list of arguments
mod(x,y)	x modulo y; remainder of integer division x/y
pow(x,y)	x to the power of y, x^y
rint(x)	Round to nearest integer.
sign(x)	Sign function: -1 if x<0; 1 if x>0.
sin(x)	sine of x
sinh(x)	hyperbolic sine of x
sqrt(x)	square root of x
tan(x)	tangent of x

Name	Description
cell(a,b)	In the context of a matrix, returns the value at row a and column b. In the context of a table, returns the value at column a and row b (remember that tables use column logic). Everywhere else, this function is undefined. In the case of tables, the column is specified as a path string; see the documentation of column() for supported path formats.
cell_(column, row)	In the context of a table, return the value of a cell specified using 1-based indexes. Wherever possible, you should use cell() instead, because inserting/removing/moving columns will break formulas using cell_().
col(c)	DEPRECATED; use column() or cell() instead. Note that the user interface still uses col() until a proper language-specific mechanism is implemented.
column("path")	In a column formula, returns the value in the given column and current row (i). The column path can either be the name of column in the current table, more generally a relative path to a column in another table (e.g. "../otherTable/col") or an absolute path (i.e., relative to the project root) e.g. "/folder/otherTable/col"). Searching for a table anywhere in the project using "otherTable/col" (without leading slash) is supported for backwards-compatibility reasons, but is strongly discouraged - a future release will drop the requirement of project-wide unique table names, at which point this usage will cease to be well-defined.
column_(index)	In a column formula, returns the value in the column given by 1-based index and current row (i). You should use column() wherever possible, because inserting/ removing/moving columns will break formulas using column_().
if(e1,e2,e3)	if e1 is true, e2 is executed else e3 is executed (works only if SciDAVis is built using muParser 1.x versions).
e1?e2:e3	if e1 is true, e2 is executed else e3 is executed (C++ style syntax; works only if SciDAVis is built using muParser >= 2.0.0).
tablecol(t,c)	DEPRECATED; use column() instead.

Table 4.3: Non-Mathematical Functions

of SciDAVis has been compiled without Python support). Next, open a Notes window and enter **`print "Hello World!"`**. Position the cursor anywhere on this line and press Ctrl+J, or select "Execute" from the context menu. The string "Hello World!" should appear below the line you entered. Congratulations, you've made contact with Python! You can also enter more complex code fragments, such as function or class definitions, in which case you have to select the whole code block before executing it.

You can also use Notes windows as a handy calculator. Enter a mathematical expression on a line of its own - say, **`5*sin(pi/2)`**. Press Ctr+Enter, or select "Evaluate" from the context menu. You are rewarded by a new line, stating (to general astonishment), that the result of evaluating your expression is `#> 5`. If you have SciPy and/or PyGSL installed, you will have immediate access to a huge number of interesting functions, browseable via the sub-menu "Functions" of the context menu. Pressing Shift+F1 while in this menu will give you a short description of the current function. The advantage of only executing/evaluating single lines or selections on request is that you can freely mix text and calculations within a Notes window.

Another particularly handy place for using Python code are column formulas. These work just like evaluating expressions in Note windows, with the additional advantage of some pre-defined variables: i (the row currently being computed), j (the column number), sr (start row), er (end row) and self (the table to which the column being computed belongs; see below for what you can do with it).

If you are already familiar with Python, you might ask yourself at this point if you can use more complicated column formulas than just single expressions (for instance, if:/else: decisions based on the current row number). The answer is: yes, you can. For the benefit of those not familiar with Python, we will give a short introduction to the language before discussing how to do this.

4.2.2 Python Basics

4.2.2.1 Expressions

Mathematical expressions work largely as expected. However, there's one caveat, especially when switching from muParser: **`a^b`** does not mean "raise a to the power of b" but rather "bitwise exclusive or of a and b"; Python's power operator is **. Thus:

```
2^3 # read: 10 xor 11 = 01
#> 1
2**3
#> 8
```

4.2.2.2 Statements

One thing you have to know when working with Python is that indentation is very important. It is used for grouping (most other languages use either braces or keywords like **`do...end`** for this). For example, executing the following:

```
x=23
for i in (1,4,5):
```

```
  x=i**2
  print(x)
```

will do what you would expect: it prints out the numbers 1, 16 and 25; each on a line of its own. Deleting just a bit of space will change the functionality of your program:

```
x=23
for i in (1,4,5):
  x=i**2
print(x)
```

will print out only one number - no, not 23, but rather 25. This example was designed to also teach you something about variable scoping: There are no block-local variables in Python.

There are two different variable scopes to be aware of: local and global variables. Unless specified otherwise, variables are local to the context in which they were defined. Thus, the variable `x` can have three different values in, say, two different Note windows and a column formula. Global variables on the other hand can be accessed from everywhere within your project. A variable `x` is declared global by executing the statement **`global x`**. You have to do this before assigning a value to `x`, but you have to do it only once within the project (no need to "import" the variable before using it). Note that there is a slight twist to these rules when you define your own functions.

The basic syntax for defining a function (for use within one particular note, for example) is

```
def answer():
  return 42
```

If you want your function to be accessible from the rest of your project, you have to declare it global before the definition:

```
global answer
def answer():
  return 42
```

You can add your own function to SciDAVis's function list. We'll also provide a documentation string that will show up, for example, in the "set column values" dialog:

```
global answer
def answer():
  "Return the answer to the ultimate  ↩
      question about life, the  ↩
      universe and everything."
  return 42
sci.mathFunctions["answer"] = answer
```

If you want to remove a function from the list, execute

```
del sci.mathFunctions["answer"]
```

Note that functions have their own local scope. That means that if you enter a function definition in a Notes window, you will not be able to access (neither reading nor writing) Notes-local variables from within the function. However, you can access global variables as usual.

If-then-else decisions are entered as follows:

```
if x>23:
  print(x)
else:
  print("The value is too small.")
```

You can do loops, too:

```
    for i in range(1, 11):
      print(i)
```

This will print out the numbers between 1 and 10 inclusively (the upper limit does not belong to the range, while the lower limit does).

4.2.3 Evaluation Reloaded

As we've already mentioned above, there's a bit more to evaluation than just expressions. Note that Python itself can indeed only evaluate expressions; the following describes a feature of SciDAVis added on top of Python in order to support more interesting column formulas.

If you use statements (e.g. variable assignments or control structures) in a formula, SciDAVis will assume it to be the body of a function. That is, the following code will not work:

```
a=1
a+a
```

The statement in the first line means that the formula cannot be evaluated as a single expression. Instead, the above code assigns every row in the column the return value of the following function:

```
def f():
  a=1
  a+a
```

However, since Python does not implicitly interpret expressions as something to return, this evaluates to nothing. The correct way to enter formulas with statements in them is to explicitly return something:

```
a=1
return a+a
```

There is a slight catch associated with this strategy. In a Notes window, SciDAVis will allow you to *evaluate* variable assignments like **`ee=1.6021773e-19`** without complaining - but this will not lead to the result presumably intended, i.e. introducing

a shortcut for the elementary charge to be used within the notes window. What actually happens is this: SciDAVis evaluates the function

```
def f():
  ee=1.6021773e-19
```

As mentioned in the Python introduction above, the function f has its own variable scope and (unless it happens to be declared global) the variable ee will only be visible within this scope (instead of the Notes window's scope). The solution is simple: always *execute* things like assignments and function definitions, never *evaluate* them.

4.2.4 Mathematical Functions

Python comes with some basic mathematical functions that are automatically imported (if you use the initialization file shipped with SciDAVis). Along with them, the constants e (Euler's number) and pi (the one and only) are defined. Many, many more functions can be obtained by installing SciPy and/or PyGSL.

4.2.5 Accessing SciDAVis's functions from Python

We will assume that you are using the initialization file shipped with SciDAVis.

4.2.5.1 Establishing contact

Accessing the objects in your project is straight-forward,

```
t = table("Table1")
m = matrix("Matrix1")
g = graph("Graph1")
n = note("Notes1")
```

as is creating new objects:

```
# create an empty table named "tony"  ↩
    with 5 rows and 2 columns:
t = newTable("tony", 5, 2)
# use defaults
t = newTable()
# create an empty matrix named "gina"  ↩
    with 42 rows and 23 columns:
m = newMatrix("gina", 42, 23)
# use defaults
m = newMatrix()
# create an empty graph window
g = newGraph()
# create an empty note named "momo"
n = newNote("momo")
# use defaults
n = newNote()
```

Name	Description
acos(x)	inverse cosine
asin(x)	inverse sine
atan(x)	inverse tangent
atan2(y,x)	equivalent to atan(y/x), but more efficient
ceil(x)	ceiling; smallest integer greater or equal to x
cos(x)	cosine of x
cosh(x)	hyperbolic cosine of x
degrees(x)	convert angle from radians to degrees
exp(x)	Exponential function: e raised to the power of x.
fabs(x)	absolute value of x
floor(x)	largest integer smaller or equal to x
fmod(x,y)	remainder of integer division x/y
frexp(x)	Returns the tuple (mantissa,exponent) such that x=mantissa*(2**exponent) where exponent is an integer and 0.5 <=abs(m)<1.0
hypot(x,y)	equivalent to sqrt(x*x+y*y)
ldexp(x,y)	equivalent to x*(2**y)
log(x)	natural (base e) logarithm of x
log10(x)	decimal (base 10) logarithm of x
modf(x)	return fractional and integer part of x as a tuple
pow(x,y)	x to the power of y; equivalent to x**y
radians(x)	convert angle from degrees to radians
sin(x)	sine of x
sinh(x)	hyperbolic sine of x
sqrt(x)	square root of x
tan(x)	tangent of x
tanh(x)	hyperbolic tangent of x

Table 4.4: Supported Mathematical Functions

New objects will always be added to the active folder. The functions table, matrix, graph and note will start searching in the active folder and, failing this, continue with a depth-first recursive search of the project's root folder. In order to access other folders, there are the functions

```
f = activeFolder()
# and
f = rootFolder()
```

which can be used to access subfolders and windows:

```
f2 = f.folders()[number]
f2 = f.folder(name, caseSensitive=True ↩
    , partialMatch=False)
t = f.table(name, recursive=False)
m = f.matrix(name, recursive=False)
g = f.graph(name, recursive=False)
n = f.note(name, recursive=False)
```

If you supply True for the recursive argument, a depth-first recursive search of all subfolders will be performed and the first match returned.

Also, every piece of code is executed in the context of an object which you can access via the `self` variable. For example, entering **`self.cell("t",i)`** as a column formula is equivalent to the convenience function **`col("t")`**.

4.2.5.2 Working with Tables

We'll assume that you have assigned some table to the variable `t`. You can access its numeric cell values with

```
t.cell(col, row)
# and
t.setCell(col, row, value)
```

Whenever you have to specify a column, you can use either the column name (as a string) or the consecutive column number (starting with 1). Row numbers also start with 1, just as they are displayed. If you want to work with arbitrary texts or the textual representations of numeric values, you can use

```
t.text(col, row)
# and
t.setText(col, row, string)
```

The number of columns and rows is accessed via

```
t.numRows()
t.numCols()
t.setNumRows(number)
t.setNumCols(number)
```

Column names can be read and written with

```
t.colName(number)
t.setColName(col, newName)
```

Normalize a single or all columns:

```
t.normalize(col)
t.normalize()
```

Import values from `file`, using `sep` as separator char and ignoring `ignore` lines at the head of the file. The flags should be self-explanatory.

```
t.importASCII(file, sep="\t", ignore ↩
    =0, renameCols=False, stripSpaces= ↩
    True, simplifySpace=False,  ↩
    newTable=False)
```

After having changed some table values from a script, you will likely want to update dependent Graphs:

```
t.notifyChanges()
```

As a simple example, let's set some column values without using the dialog.

```
t = table("table1")
for i in range(1, t.numRows()+1):
  t.setCell(1, i, i**2)
t.notifyChanges()
```

4.2.5.3 Working with Matrices

We'll assume that you have assigned some matrix to the variable `m`. Accessing cell values is very similar to Table, but since Matrix doesn't use column logic, row arguments are specified before columns and obviously you can't use column name.

```
m.cell(row, col)
m.setCell(row, col, value)
m.text(row, col)
m.setText(row, col, string)
```

Also like with tables, there's

```
m.numRows()
# and
m.numCols()
```

4.2.5.4 Plotting and Working with Graphs

If you want to create a new Graph window for some data in table table1, you can use the plot command:

```
g = plot(table, column, type)
```

`type` is a number between 0 and 10 and specifies the desired plot type:

0. Line

1. Symbols

2. Line and Symbols

3. Columns

4. Area

5. Pie

6. Vertical drop lines

7. Splines and Symbols

8. Vertical steps

9. Histogram

10. Rows

You can plot more than one column at once by giving a Python tuple (see the Python Tutorial) as an argument:

```
g = plot(table("table1"), (2,4,7), 2)
```

If you want to add a curve to an existing Graph window, you have to choose the destination layer. Usually,

```
l = g.activeLayer()
```

will do the trick, but you can also select a layer by its number:

```
l = g.layer(num)
```

You can then add or remove curves to or from this layer:

```
l.insertCurve(table, column, type=1)
l.insertCurve(table, Xcolumn, Ycolumn, ↩
     type=1)
l.removeCurve(curveName)
l.removeCurve(curveNumber)
l.deleteFitCurves()
```

In case you need the number of curves on a layer, you can get it with

```
l.numCurves()
```

Use the following functions to change the axis attachment of a curve:

```
c = l.curve(number)
l.setCurveAxes(number, x-axis, y-axis)
c.setXAxis(x-axis)
c.setYAxis(y-axis)
```

where `number` is curve's index, `x-axis` is either `Layer.Bottom` or `Layer.Top` and `y-axis` is either `Layer.Left` or `Layer.Right`

```
l.setCurveAxes(0, Layer.Top, Layer. ↩
    Right)  # modify the first curve  ↩
    in the layer (curve index is 0)
c.setXAxis(Layer.Bottom)
c.setYAxis(Layer.Right)  # attach  ↩
    curve c to the right Y axis
```

Layers and whole Graphs can be printed and exported from within Python. Before you do this, you would probably want to change layer and axis titles as well as legend texts:

```
l.setTitle(title)
l.setXTitle(Xtitle)
l.setYTitle(Ytitle)
l.setLegend(text)
```

You can also customize the scales of the different axes using:

```
l.setScale(int axis, double start,  ↩
    double end, double step=0.0, int  ↩
    majorTicks=5, int minorTicks=5,  ↩
    int type=0, bool inverted=False);
```

where `axis` is one of `Layer.Left`, `Layer.Right`, `Layer.Bottom`, `Layer.Top`, type specifies the desired scale type: `0` for `Linear` or `1` for `Log10` and `step` defines the size of the interval between the major scale ticks. If not specified (default value is 0.0), the step size is calculated automatically. The other flags should be self-explanatory.

Now, here is how you can export a layer

```
l.print()
l.exportToSVG(filename)
l.exportToEPS(filename)
l.exportImage(filename, filetype="PNG ↩
    ", quality=100, transparent=False)
```

and a graph

```
g.print()
g.exportToSVG(filename)
g.exportToEPS(filename)
```

4.2.5.5 Fitting

Assuming you have a Graph named "graph1" with a curve entitled "table1_2" (on its active layer), a minimal Fit example would be:

```
f = GaussFit(graph("graph1"). ↩
    activeLayer(), "table1_2")
f.guessInitialValues()
f.fit()
```

This creates a new GaussFit object on the curve, lets it guess the start parameters and does the fit. The following fit types are supported:

- LinearFit(layer, curve)
- PolynomialFit(layer, curve, degree=2, legend=False)
- ExponentialFit(layer, curve, growth=False)
- TwoExpFit(layer, curve)
- ThreeExpFit(layer, curve)
- GaussFit(layer, curve)
- GaussAmpFit(layer, curve)
- LorentzFit(layer,curve)
- SigmoidalFit(layer, curve)
- NonLinearFit(layer, curve)

  ```
  f = NonLinearFit(layer, curve)
  f.setParameters(name1, ...)
  f.setFormula(formula_string)
  ```

- PluginFit(layer, curve)

  ```
  f = PluginFit(layer, curve)
  f.load(pluginName)
  ```

For each of these, you can optionally restrict the X range that will be used for the fit, like in

```
f = LinearFit(graph("graph1"). ↩
    activeLayer(), "table1_2", 2, 7)
f.fit()
```

After creating the Fit object and before calling its fit() method, you can set a number of parameters that influence the fit:

```
f.setDataFromCurve(curve)     change data source
f.setDataFromCurve(curve, graph)    change data source
f.setDataFromCurve(curve, from, to)   change data source
f.setDataFromCurve(curve, from, to, graph)  change data source
f.setInterval(from, to)       change data range
f.setInitialValue(number, value)
f.setInitialValues(value1, ...)
f.guessInitialValues()
f.setAlgorithm(algo) # algo = Fit.ScaledLevenbergMarquardt, Fit. ↩
    UnscaledLevenbergMarquardt, Fit.NelderMeadSimplex


f.setYErrorSource(ErrorSource, colname) # ErrorSource can be: 0 / Fit. ↩
    UnknownErrors, 1 / Fit.AssociatedErrors,
2 / Fit.PoissonErrors, or 3 / Fit.CustomErrors
f.setTolerance(tolerance)
f.setOutputPrecision(precision)
f.setMaximumIterations(number)
f.scaleErrors(yes = True)
f.setColor(qt.QColor("green"))      change the color of the result fit curve  ↩
    to green (default color is red)
```

After you've called fit(), you have a number of possibilities for extracting the results:

```
f.results()
f.errors()
f.chiSquare()
f.rSquare()
f.dataSize()
f.numParameters()
f.parametersTable("params")
f.covarianceMatrix("cov")
```

4.2.6 API documentation

Classes mentioned below that start with "Q" mostly belong to Qt and can be used via PyQt (with the exception of classes starting with "Qwt", which belong to Qwt). If you want to know what you can do with such classes (e.g. QIcon or QDateTime), see the PyQt reference documentation. It's also useful to know that you can call any QWidget method exported by PyQt on instances of MDIWindow (and thus Table, Graph, Matrix and Note).

4.2.6.1 class AbstractAspect (inherits QObject)

A base class for content of a project. While currently the only AbstractAspect accessible to you is Column, a future release will make more extensive usage of this; probably

with some revisions in the design, so only some basic functions are documented (and supported) for now.

```
col = newTable("my-table",2,30).column("1")
col.setName("abc")
table("my-table").column("2").remove()
```

index() Return the 0-based index of the Aspect in its sibling list.

name() Return the Aspect's name.

setName(string) Change the Aspect's name.

comment() Return comment attached to the Aspect.

setComment(string) Change the comment attached to the Aspect.

icon() Return the icon (a QIcon) used to designate the Aspect's type; for columns, this is the data type icon in the table header.

creationTime() Return point in time the Aspect was created by the user (as a QDateTime).

remove() Remove Aspect from the project (can be undone using Edit->Undo menu).

signal void aspectDescriptionAboutToChange(const AbstractAspect*) Emitted before the description (name or comment) is changed.

signal void aspectDescriptionChanged(const AbstractAspect*) Emitted after the description (name or comment) has changed.

signal void aspectAboutToBeRemoved(const AbstractAspect*) Emitted before the Aspect is removed.

4.2.6.2 class Column (inherits AbstractAspect)

Represents a column in a table.

In addition to the valueAt()/setValueAt() interface described below, starting with SciDAVis 0.2.4, Y columns support getting/setting row values via the values in the corresponding X column. This is done using Python's item access notation for sequence types (as in col["abc"]).

```
# basic access
tab = newTable("tabula1",2,20)
col = tab.column("1")
col.setValueAt(1, 123.0)
print col.valueAt(1)
# replacing multiple values at once is more efficient than setting them one at ↩
    a time
col.replaceValues(5, [11.2, 12.3, 23.4, 34.5, 45.6])
# set a value in the second column based on the content of the first one
```

```
# (needs SciDAVis >= 0.2.4)
tab.column("2")[23.4] = 46.8
dest = table("tabula1").column("2")
# also, copying from another column can be done in one go:
dest.copy(col, 5, 2, 10)
```

columnMode() A string designating the data type; one of "Numeric", "Text", "Month", "Day" or "DateTime".

setColumnMode(string) Change the column's data type; argument must be one of "Numeric", "Text", "Month", "Day" or "DateTime".

copy(Column) Copy complete content of other column, which must have the same data type (mode). Returns a boolean indicating success or failure.

copy(Column,int,int,int) copy(source, source_start, dest_start, num_rows) copies num_rows values from source, starting to read at row source_start and to write at row dest_start.

rowCount() Number of rows in the column.

insertRows(int, int) insertRows(before, count) insert count empty rows before row numbered before

removeRows(int, int) removeRows(first, count) removes count rows starting with row numbered first

plotDesignation() Return a string representing the plot designation of the column; one of "noDesignation", "X", "Y", "Z", "xErr" or "yErr".

setPlotDesignation(string) Set the plot designation. The argument must be one of "noDesignation", "X", "Y", "Z", "xErr" or "yErr".

clear() Clear content of all cells in the column, marking them as empty/invalid.

isInvalid(int) Returns boolean indicating whether the given row is marked as empty/invalid.

formula(int) Return formula used to compute the cell value in the given row.

setFormula(int, string) Sets formula for the given row to string.

clearFormulas() Discard all formulas associated with cells.

valueAt(int) Retrieve value of the specified cell (given by 0-based index), assuming that the column has columnMode() == "Numeric".

setValueAt(int, double) Set value of specified cell (given by 0-based index), assuming that the column has columnMode() == "Numeric".

replaceValues(int, list of double values) Mass-update cells starting at the indicated row, assuming that the column's columnMode() is one of "Numeric". Compared with setValueAt(), this has the advantage of being much more efficient; particularly if the column has dependant plots.

textAt(int) Retrieve value of the specified cell (given by 0-based index), assuming that the column has columnMode() == "Text".

setTextAt(int, string) Set value of specified cell (given by 0-based index), assuming that the column has columnMode() == "Text".

replaceTexts(int, list of strings) Mass-update cells starting at the indicated row, assuming that the column has columnMode() == "Text". Compared with setTextAt(), this has the advantage of being much more efficient; particularly if the column has dependant plots.

dateAt(int) Retrieve value of the specified cell (given by 0-based index), assuming that the column's columnMode() is one of "Month", "Day", "DateTime".

setDateAt(int, QDate) Set value of specified cell (given by 0-based index), assuming that the column's columnMode() is one of "Month", "Day", "DateTime".

timeAt(int) Retrieve value of the specified cell (given by 0-based index), assuming that the column has columnMode() == "DateTime".

setTimeAt(int, QTime) Set value of specified cell (given by 0-based index), assuming that the column has columnMode() == "DateTime".

dateTimeAt(int) Retrieve value of the specified cell (given by 0-based index), assuming that the column's columnMode() is one of "Month", "Day", "DateTime".

setDateTimeAt(int, QDateTime) Set value of specified cell (given by 0-based index), assuming that the column's columnMode() is one of "Month", "Day", "DateTime".

replaceDateTimes(int, list of QDateTime values) Mass-update cells starting at the indicated row, assuming that the column's columnMode() is one of "Month", "Day", "DateTime". Compared with setDateTimeAt(), this has the advantage of being much more efficient; particularly if the column has dependant plots.

x() Returns the X Column associated with this (Y, xErr or yErr) column.

y() Returns the Y Column associated with this (xErr or yErr) column, or the first Y column associated with this X column.

4.2.6.3 class MDIWindow (inherits QWidget)

Base class of Table, Matrix, Graph and Note. A redesigned analogue under the name of "Part" (inheriting from AbstractAspect) is under development; it should be possible to provide a backwards-compatible interface, though.

```
tmp = newTable("temp1", 2, 10)
tmp.setWindowLabel("a temporary table")
tmp.setCaptionPolicy(MDIWindow.Label)
# ...
tmp.confirmClose(False)
tmp.close()
```

name() Return the window's name (added in SciDAVis 0.2.3).

setName(string) Set window's name. IMPORTANT: This was added in SciDAVis 0.2.3, but unfortunately is BROKEN in this release. Usable starting with SciDAVis 0.2.4.

windowLabel() Returns the comment associated with the window (yes, the confusing name will be fixed in a future release).

setWindowLabel(string) Set comment associated with the window (yes, the confusing name will be fixed in a future release).

captionPolicy() Return caption policy (i.e., what to display in the title bar). One of the integer constants MDIWindow.Name, MDIWindow.Label or MDIWindow.Both.

setCaptionPolicy(int) Set caption policy (i.e., what to display in the title bar). Must be one of the integer constants MDIWindow.Name, MDIWindow.Label or MDIWindow.Both.

folder() Return the folder this window belongs to (an object of class Folder).

confirmClose(boolean) Set a flag indicating whether the user should be given the opportunity to cancel removing the window or minimize it instead when trying to close it. When False, close() will remove the window without asking for confirmation, which is useful for temporary objects created by a script.

clone() Returns a copy of this window. Added in SciDAVis 0.2.4.

4.2.6.4 class Table (inherits MDIWindow)

Class of table (spreadsheet) windows.

```
tab = newTable("tabula",2,10)
for j in range(0, tab.numCols()):
    for i in range(0, tab.numRows()):
        tab.column(j).setValueAt(i,i*j)
tab.normalize()
```

numRows() Returns the number of rows of the table.

numCols() Returns the number of columns of the table.

setNumRows(int) Sets the number of rows of the table.

setNumCols(int) Sets the number of columns of the table.

column(string or int) Returns Column indicated by name (preferred) or 0-based index. For the latter form, keep in mind that inserting, removing or moving columns will break scripts which refer to a specific column by index. Starting with SciDAVis 0.2.4, columns may also be accessed using the [] operator, i.e. using table("table name")[column name or index].

text(string or int, int) DEPRECATED. Use column(col).textAt(row) or str(column(col).valueAt(row)) instead (depending on the column's mode). CAVEAT: text() indexes are 1-based, while column() and textAt() use more conventional 0-based indexes!

cell(string or int, int) DEPRECATED. Use column(col).valueAt(row) instead. CAVEAT: cell() indexes are 1-based, while column() and valueAt() use more conventional 0-based indexes!

setText(string or int, int, string DEPRECATED. Use column(col).setTextAt(row,value) instead. CAVEAT: setText() indexes are 1-based, while column() and setTextAt() use more conventional 0-based indexes!

setCell(string or int, int, double DEPRECATED. Use column(col).setValueAt(row,value) instead. CAVEAT: setCell() indexes are 1-based, while column() and setValueAt() use more conventional 0-based indexes!

colName(int) DEPRECATED. Use column(col).name() instead.

setColName(int,string) DEPRECATED. Use column(col).setName(name) instead.

setComment(int,string) DEPRECATED. Use column(col).setComment(text) instead.

setCommand(string or int, string) Set formula indicated by first argument to the text given in the second argument.

notifyChanges() DEPRECATED. Update notifications are now done automatically.

importASCII(string, string, int, bool, bool, bool) importASCII(file, separator="\t", ignored_lines=0, rename_cols=False, strip_spaces=True, simplify_spaces=False) imports file into this table, skipping ignored_lines lines at the beginning and splitting the rest into columns according to the given separator and flags.

exportASCII(string, string, bool, bool) exportASCII(file, separator="\t", with_comments=False, selection_only=False) exports this table to the given file with the column separator given in the second argument; optionally including column comments and optionally exporting only selected cells.

normalize(string or int) Normalize specified column to a maximum cell value of one.

normalize() Normalize all columns in table to a maximum cell value of one.

sortColumn(string or int, int=0) Sort column indicated in the first argument. The second argument selects the sorting order; 0 means ascending, 1 means descending.

sort(int=0, int=0, string='''') sort(type, order, leading_column) sorts all columns in the table either separately (type=0) or together (type=1) with the leading column given by name. The second argument selects the sorting order; 0 means ascending, 1 means descending.

sortColumns(list of strings, int, int, string) sortColumns(columns, type=0, order=0, leading_column="") sorts columns given as a tuple of names either separately (type=0) or together (type=1) with the leading column given by name. The third argument selects the sorting order; 0 means ascending, 1 means descending.

4.2.6.5 class Matrix (inherits MDIWindow)

Class of matrix windows.

```
mat = newMatrix("mat", 10, 20)
mat.setCoordinates(100,2000,100,1000)
mat.setFormula("i*j")
mat.recalculate()
```

numRows() Returns the number of rows in the matrix.

numCols() Returns the number of columns in the matrix.

setNumRows(int) Changes the number of rows in the matrix.

setNumCols(int) Changes the number of columns in the matrix.

setDimensions(int,int) setDimensions(rows, cols) changes the number of rows and columns simultaneously.

cell(int,int) cell(row,column) returns the content of the indicated cell as a number.

text(int,int) text(row,column) returns the content of the indicated cell as its textual representation.

setCell(int,int,double) setCell(row,column,value) changes the content of the indicated cell to the indicated numeric value.

setText(int,int,string) setText(row,column,value) interprets the string value according to the numeric format of the matrix and changes the content of the indicated cell accordingly.

xStart() Returns logical X coordinate of the first column.

xEnd() Returns logical X coordinate of the last column.

yStart() Returns logical Y coordinate of the first row.

yEnd() Returns logical Y coordinate of the last row.

setCoordinates(double,double,double,double) setCoordinates(x_start, x_end, y_start, y_end) changes the logical coordinate system associated with the matrix.

setFormula(string) Changes the formula used to (re)calculate cell values of the matrix.

recalculate() Recalculate all cell values from the currently set formula. Returns boolean indicating success or failure. Added in SciDAVis 0.2.4.

setNumericPrecision(int) Changes the number of decimal digits being displayed.

transpose() Mirror matrix at main diagonal (aka matrix transposition).

invert() Replace the content of the matrix by its inverse (with respect to matrix multiplication).

determinant() Returns determinant of the matrix.

4.2.6.6 class ArrowMarker

A line or an arrow displayed on a graph.

```
arrow = ArrowMarker()
arrow.setStart(50,200)
arrow.setEnd(400,400)
arrow.setWidth(2)
arrow.drawStartArrow(False)
arrow.drawEndArrow(True)
arrow.setColor(Qt.green)

layer = newGraph().activeLayer()
layer.addArrow(arrow)
layer.replot()
```

ArrowMarker() Creates a new arrow marker. You can then set various attributes (see below) and hand it to Layer.addArrow().

setStart(double, double) setStart(x,y) sets the line's start point in plot coordinates (i.e., as displayed on the axes).

setEnd(double,double) setEnd(x,y) sets the line's end point in plot coordinates (i.e., as displayed on the axes).

setStyle(Qt.PenStyle) Sets pen style used to draw the line and arrow heads. One of Qt.NoPen, Qt.SolidLine, Qt.DashLine, Qt.DotLine, Qt.DashDotLine, Qt.DashDotDotLine.

setColor(QColor) Sets the line/arrow head color. Most commonly, the argument will be either a basic color (Qt.white, Qt.black, Qt.red, Qt.darkRed etc. - see Qt.GlobalColor for details) or a RGBA spec in the form QtGui.QColor(red, green, yellow, alpha) with channels given as integers in the range [0,255] (see QColor for details).

setWidth(int) Sets the pen width used to draw line and arrow heads.

drawStartArrow(bool=True) Sets whether an arrow head is drawn at the line's start.

drawEndArrow(bool=True) Sets whether an arrow head is drawn at the line's end.

setHeadLength(int) Sets the length of the arrow heads.

setHeadAngle(int) Sets the angle defining the "sharpness" of the arrow heads.

fillArrowHead(bool) Sets whether arrow heads are to be filled out.

4.2.6.7 class ImageMarker

An image to be displayed on a graph.

```
layer = newGraph().activeLayer()
image = layer.addImage("/usr/share/icons/hicolor/128x128/apps/scidavis.png")
image.setCoordinates(200,800,700,300)
layer.replot()
```

ImageMarker(string) Creates a new image marker displaying the content of the specified image file.

fileName() Returns the name of the file the image marker refers to.

size() Returns the size the image takes up in paint (pixel) coordinates as a QSize.

setSize(int,int) Sets the size the image takes up in paint (pixel) coordinates.

setCoordinates(double,double,double,double) setCoordinates(left, top, right, bottom) changes position and size of the image marker in plot coordinates.

4.2.6.8 class Legend

Text boxes displayed on a graph. While this is also used for legends, a better (since more general) name would probably be TextMarker.

```
layer = newGraph().activeLayer()
legend = layer.newLegend("hello world")
legend.setBackgroundColor(Qt.green)
legend.setTextColor(Qt.darkBlue)
legend.setFont(QtGui.QFont("Arial",14,QtGui.QFont.Bold))
layer.replot()
```

setText(string) Changes the legend's content.

setTextColor(QColor) Changes the text color. Most commonly, the argument will be either a basic color (Qt.white, Qt.black, Qt.red, Qt.darkRed etc. - see Qt.GlobalColor for details) or a RGBA spec in the form QtGui.QColor(red, green, yellow, alpha) with channels given as integers in the range [0,255] (see QColor for details).

setFrameStyle(int) Sets the style of frame drawn around the text. 0 for none, 1 for line or 2 for line with shadow.

setBackgroundColor(QColor) Changes the background color. Most commonly, the argument will be either a basic color (Qt.white, Qt.black, Qt.red, Qt.darkRed etc. - see Qt.GlobalColor for details) or a RGBA spec in the form QtGui.QColor(red, green, yellow, alpha) with channels given as integers in the range [0,255] (see QColor for details).

setFont(QFont) Sets the font used to render the text. See QFont documentation for how to construct a valid argument.

setOriginCoord(double,double) setOriginCoord(x,y) sets the position of the top-left corner in plot coordinates.

4.2.6.9 class QwtSymbol

Represents a type of symbol on a scatter plot (ellipse, rectangle etc.) together with attributes determining parameters like color, size etc. This class is part of the Qwt library, but exported and documented as part of SciDAVis's API for simplicity. Also, convenience methods for setting outline/fill colors are added on top of Qwt.

```
symbol = QwtSymbol()
symbol.setStyle(QwtSymbol.Triangle)
symbol.setOutlineColor(QtGui.QColor(Qt.red))
symbol.setFillColor(QtGui.QColor(Qt.green))
symbol.setSize(20)
# assuming Graph1 exists and contains a plot
layer = graph("Graph1").activeLayer()
layer.curve(0).setSymbol(symbol)
layer.replot()
```

QwtSymbol() Construct new symbol with default settings (= no symbol).

QwtSymbol(Style, QBrush, QPen, QSize) Construct new symbol, with the four properties set as if given to setStyle(), setBrush(), setPen() and setSize(), respectively.

setColor(QColor) Simultaneously sets color for filling and drawing the outline. Modifies pen() and brush(). Note that due to the setColor(int) overload, and unlike other methods accepting QColor arguments, basic colors need to be explicitly specified as QtGui.QColor(Qt.white) etc. instead of just Qt.white. As usual, it's also possible to give an RGBA spec in the form QtGui.QColor(red, green, yellow, alpha) with channels given as integers in the range [0,255] (see QColor for details).

setColor(int) Convenience overload of setColor(QColor) choosing the color to be set from the palette used by SciDAVis for automatically assigning colors to new curves.

setOutlineColor(QColor) Sets the color used for drawing the outline of the symbol. Modifies pen(). Note that due to the setOutlineColor(int) overload, and unlike other methods accepting QColor arguments, basic colors need to be explicitly specified as QtGui.QColor(Qt.white) etc. instead of just Qt.white. As usual, it's also possible to give an RGBA spec in the form QtGui.QColor(red, green, yellow, alpha) with channels given as integers in the range [0,255] (see QColor for details).

setOutlineColor(int) Convenience overload of setOutlineColor(QColor) choosing the color to be set from the palette used by SciDAVis for automatically assigning colors to new curves.

setFillColor(QColor) Sets the color used for filling the interior of the symbol. Modifies brush(). Note that due to the setFillColor(int) overload, and unlike other methods accepting QColor arguments, basic colors need to be explicitly specified as QtGui.QColor(Qt.white) etc. instead of just Qt.white. As usual, it's also possible to give an RGBA spec in the form QtGui.QColor(red, green, yellow, alpha) with channels given as integers in the range [0,255] (see QColor for details).

setFillColor(int) Convenience overload of setFillColor(QColor) choosing the color to be set from the palette used by SciDAVis for automatically assigning colors to new curves.

clone() Returns an independent copy of the symbol object.

setSize(QSize) Sets size of the symbol in paint (pixel) coordinates.

setSize(int,int=-1) setSize(width,height) sets the size of the symbol in paint (pixel) coordinates. If height=-1 (default), it is set to equal to width.

setBrush(QBrush) Sets the brush used to fill the interior of the symbol. See PyQt documentation for how to construct a QBrush.

setPen(QPen) Sets the pen used to draw the border of the symbol. See PyQt documentation for how to construct a QPen.

setStyle(Style) Sets the type of symbol to draw. The argument needs to be one of the integer constants QwtSymbol.NoSymbol, QwtSymbol.Ellipse, QwtSymbol.Rect, QwtSymbol.Diamond, QwtSymbol.Triangle, QwtSymbol.DTriangle, QwtSymbol.UTriangle, QwtSymbol.LTriangle, QwtSymbol.RTriangle, QwtSymbol.Cross, QwtSymbol.XCross, QwtSymbol.HLine, QwtSymbol.VLine, QwtSymbol.Star1, QwtSymbol.Star2, QwtSymbol.Hexagon.

brush() Returns the currently set brush (a QBrush) for filling the interior of the symbol. Due to a design limitation of Qwt, setting attributes of the brush directly has no effect. You need to create a copy of the brush, change its attributes and hand it again to setBrush().

pen() Returns the currently set pen (a QPen) for drawing the border of the symbol. Due to a design limitation of Qwt, setting attributes of the pen directly has no effect. You need to create a copy of the pen, change its attributes and hand it again to setPen().

size() Returns the currently set size (a QSize) in paint (pixel) coordinates for drawing the symbol. Due to a design limitation of Qwt, setting attributes of the QSize directly has no effect. You need to create a copy of the size, change its attributes and hand it again to setSize().

style() Returns an integer denoting the currently set type of symbol. See setStyle() for possible values.

4.2.6.10 class QwtPlotCurve

Represents a curve with symbols and/or lines on a graph. This class is part of the Qwt library, but exported and documented as part of SciDAVis's API for simplicity. Also, convenience methods for setting outline/fill colors and fill style are added on top of Qwt.

```
# assuming Graph1 exists and contains a lines plot
layer = graph("Graph1").activeLayer()
curve = layer.curve(0)
curve.setOutlineColor(QtGui.QColor(Qt.red))
curve.setFillColor(QtGui.QColor(Qt.green))
curve.setFillStyle(Qt.CrossPattern)
layer.replot()
```

dataSize() Returns the number of points in the curve.

x(int) Returns X coordinate of indicated point.

y(int) Returns Y coordinate of indicated point.

minXValue() Return smallest X value of the curve's points.

maxXValue() Return largest X value of the curve's points.

minYValue() Return smallest Y value of the curve's points.

maxYValue() Return largest Y value of the curve's points.

setXAxis(x-axis) Set the x-axis to which the curve is attached. x-axis is Layer.Bottom or Layer.Top.See also: setCurveAxes.

setYAxis(y-axis) Set the x-axis to which the curve is attached. y-axis is Layer.Left or Layer.Right.See also: setCurveAxes.

setPen(QPen) Sets the pen used to draw the lines of the curve. See PyQt documentation for how to construct a QPen.

pen() Returns the currently set pen (a QPen) for drawing the lines of the curve. Due to a design limitation of Qwt, setting attributes of the pen directly has no effect. You need to create a copy of the pen, change its attributes and hand it again to setPen().

setBrush(QBrush) Sets the brush used to fill the area under the curve. See PyQt documentation for how to construct a QBrush.

brush() Returns the currently set brush (a QBrush) for filling the area under the curve. Due to a design limitation of Qwt, setting attributes of the brush directly has no effect. You need to create a copy of the brush, change its attributes and hand it again to setBrush().

setSymbol(QwtSymbol) Specifies whether and how symbols are drawn. See class QwtSymbol.

symbol() Returns symbol parameters currently in effect. See class QwtSymbol. Due to a design limitation of Qwt, setting attributes of the symbol directly has no effect. You need to create a copy of the symbol, change its attributes and hand it again to setSymbol().

setColor(QColor) Simultaneously sets color for drawing lines and filling the area under the curve. Modifies pen() and brush(). Note that due to the setColor(int) overload, and unlike other methods accepting QColor arguments, basic colors need to be explicitly specified as QtGui.QColor(Qt.white) etc. instead of just Qt.white. As usual, it's also possible to give an RGBA spec in the form QtGui.QColor(red, green, yellow, alpha) with channels given as integers in the range [0,255] (see QColor for details).

setColor(int) Convenience overload of setColor(QColor) choosing the color to be set from the palette used by SciDAVis for automatically assigning colors to new curves.

setOutlineColor(QColor) Sets the color used for drawing the lines of the curve. Modifies pen(). Note that due to the setOutlineColor(int) overload, and unlike other methods accepting QColor arguments, basic colors need to be explicitly specified as QtGui.QColor(Qt.white) etc. instead of just Qt.white. As usual, it's also possible to give an RGBA spec in the form QtGui.QColor(red, green, yellow, alpha) with channels given as integers in the range [0,255] (see QColor for details).

setOutlineColor(int) Convenience overload of setOutlineColor(QColor) choosing the color to be set from the palette used by SciDAVis for automatically assigning colors to new curves.

setFillColor(QColor) Sets the color used for filling the area under the curve. Modifies brush(). Note that due to the setFillColor(int) overload, and unlike other methods accepting QColor arguments, basic colors need to be explicitly specified as QtGui.QColor(Qt.white) etc. instead of just Qt.white. As usual, it's also possible to give an RGBA spec in the form QtGui.QColor(red, green, yellow, alpha) with channels given as integers in the range [0,255] (see QColor for details).

setFillColor(int) Convenience overload of setFillColor(QColor) choosing the color to be set from the palette used by SciDAVis for automatically assigning colors to new curves.

setFillStyle(Qt.BrushStyle) Set pattern used for filling the area under the curve. Argument must be one of Qt.SolidPattern, Qt.Dense1Pattern, Qt.Dense2Pattern, Qt.Dense3Pattern, Qt.Dense4Pattern, Qt.Dense5Pattern, Qt.Dense6Pattern, Qt.Dense7Pattern, Qt.NoBrush, Qt.HorPattern, Qt.VerPattern, Qt.CrossPattern, Qt.BDiagPattern, Qt.FDiagPattern, Qt.DiagCrossPattern, Qt.LinearGradientPattern, Qt.RadialGradientPattern, Qt.ConicalGradientPattern. See PyQt documentation for visual overview and more information.

4.2.6.11 class Grid

Handles options related to the grid drawn on a Layer. Added in SciDAVis 0.2.4.

```
layer = newGraph().activeLayer()
layer.showGrid()
layer.grid().setMajorPen(QtGui.QColor(Qt.black))
layer.replot()
```

setMajor(bool) Enables/disables drawing of major grid lines for both axes.

setXMajor(bool) Enables/disables drawing of major grid lines for X axis.

setYMajor(bool) Enables/disables drawing of major grid lines for Y axis.

xMajor() Boolean indicating whether major grid lines for X axis are enabled.

yMajor() Boolean indicating whether major grid lines for Y axis are enabled.

setMinor(bool) Enables/disables drawing of minor grid lines for both axes.

setXMinor(bool) Enables/disables drawing of minor grid lines for X axis.

setYMinor(bool) Enables/disables drawing of minor grid lines for Y axis.

xMinor() Boolean indicating whether minor grid lines for X axis are enabled.

yMinor() Boolean indicating whether minor grid lines for Y axis are enabled.

setXZeroLine(bool) Enables/disables drawing of line at X=0.

xZeroLine() Boolean indicating whether a line is drawn at X=0.

setYZeroLine(bool) Enables/disables drawing of line at Y=0.

yZeroLine() Boolean indicating whether a line is drawn at Y=0.

setMajorPen(QPen) Sets color, line width, line style etc. of major grid lines. See PyQt reference for class QPen.

setXMajorPen(QPen) Sets color, line width, line style etc. of major grid lines for X axis. See PyQt reference for class QPen.

setYMajorPen(QPen) Sets color, line width, line style etc. of major grid lines for Y axis. See PyQt reference for class QPen.

xMajorPen() Returns QPen used for drawing major grid lines for X axis. See PyQt reference for class QPen.

yMajorPen() Returns QPen used for drawing major grid lines for Y axis. See PyQt reference for class QPen.

setMinorPen(QPen) Sets color, line width, line style etc. of minor grid lines. See PyQt reference for class QPen.

setXMinorPen(QPen) Sets color, line width, line style etc. of minor grid lines for X axis. See PyQt reference for class QPen.

setYMinorPen(QPen) Sets color, line width, line style etc. of minor grid lines for Y axis. See PyQt reference for class QPen.

xMinorPen() Returns QPen used for drawing minor grid lines for X axis. See PyQt reference for class QPen.

yMinorPen() Returns QPen used for drawing minor grid lines for Y axis. See PyQt reference for class QPen.

4.2.6.12 class Layer (inherits QWidget)

A layer on a graph. All elements in a graph are organized in layers, so whenever you're working with graphs, you're also dealing with layers. Note that many changes do not show up until you call replot() - if you're changing many options on a complex layer, this is faster than automatically updating the layer on every change.

```
layer = newGraph().activeLayer()
layer.setTitle("Murphy Certainty Principle")
layer.setXTitle("time")
layer.setYTitle("motivation")
layer.insertFunctionCurve("1/x", 0, 10)
# the constants QwtPlot.xBottom (=2) and QwtPlot.yLeft (=0) were added in ↩
    SciDAVis 0.2.4
layer.setScale(QwtPlot.xBottom, 0, 10)
layer.setScale(QwtPlot.yLeft, 0, 10)
layer.setBackgroundColor(QtGui.QColor(18,161,0))
layer.setCanvasColor(QtGui.QColor(161,120,50))
layer.curve(0).setPen(QtGui.QPen(Qt.yellow, 3))
layer.removeLegend()
layer.replot()
```

isPiePlot() Boolean indicating whether this layer is a pie plot. Pie plots are always on a separate layer.

pieLegend() Returns content of legend, assuming this layer is a pie plot. See isPiePlot().

insertCurve(Table, string, int, int) insertCurve(table, column, style=1, color=-1) plots the data of the specified Y column and the corresponding designated X column, optionally specifying style and color, and returning a boolean indicating success or failure. Color is an index in the palette used by SciDAVis for automatically assigning colors to new curves. Style is one of the following codes:

0 Line

1 Symbols

2 Line and Symbols

3 Columns

4 Area

5 Pie

6 Vertical drop lines

7 Splines and Symbols

8 Vertical steps

9 Histogram

10 Rows

insertCurve(Table, string, string int, int) insertCurve(table, x_column, y_column style=1, color=-1) works like the other insertCurve() variant above, but allows you to explicitly specify the X column you want to plot against instead of determining it by designation.

insertFunctionCurve(string, double=0, double=1, int=100, string=QtCore.QString()) insertFunctionCurve(formula, start, end, points, title) inserts a function curve specified by formula, which is evaluated using muParser with abscissa "x", and returns a boolean indicating success or failure. Optionally, the interval [start,end] for the evaluation, the number of points where the function will be evaluated and the curve title can be given. Added in SciDAVis 0.2.4.

insertPolarCurve(string, string, double=0, double=2*pi, string="t", int=100, string=QtCore.QString()) insertPolarCurve(radial, angular, start, end, parameter, points, title) inserts a function curve specified by formulas for the radial and angular components in polar coordinates, both of which are evaluated using muParser with given parameter ("t" by default), and returns a boolean indicating success or failure. Optionally, the interval [start,end] covered by the free parameter, the number of points where the function will be evaluated and the curve title can be overridden. Added in SciDAVis 0.2.4.

insertParametricCurve(string, string, double=0, double=1, string="t", int=100, string=QtCore.QString()) insertParametricCurve(x_formula, y_formula, from, to, parameter, points, title) insert a function curve specified by formulas for the abscissa and ordinate components in cartesian coordinates, both of which are evaluated using muParser with given parameter ("t" by default), and returns a boolean indicating success or failure. Optionally, the interval [start,end] covered by the free parameter, the number of points where the function will be evaluated and the curve title can be overridden. Added in SciDAVis 0.2.4.

addErrorBars(string, Table, string, int=1, int=1, int=8, QColor=QColor(Qt.black), bool=True, bool=True, bool=T addErrorBars(curve, table, err_col_name, orientation, width, cap_length, color, through, minus, plus) adds error bars to the curve specified by its name (as displayed in the add/remove curve dialog and the curves' context menu). Data for the error bars is taken from the given table and column. Optionally, you can override the orientation (default is 1 for vertical, 0 means horizontal), the width of the pen used to draw the error bars, the length of the caps in pixels, the color of the pen used to draw the error bars, and flags indicating whether the error bars' line strikes through the underlying curve, and whether the minus and plus side of the error bars are to be drawn.

removeCurve(int or string) Remove curve specified by index (in the range [0,numCurves()-1]) or title.

deleteFitCurves() Remove all curves that show the result of a fit operation.

numCurves() Returns the number of curves plotted on this layer.

showCurve(int,bool) Show/hide the curve specified by its index (in the range [0,numCurves()-1]). Hidden curves are not plotted, but otherwise remain part of the Layer; they keep their legend item, count as part of numCurves(), can be accessed by curve() etc.

curve(int or string) Access curve (as QwtPlotCurve instance) specified by its title (as displayed in the add/remove curves dialog and the curve's context menu) or index (integer in the range [0,numCurves()-1].

curves() Returns a list of all curves in the layer. Added in SciDAVis 0.2.4.

addArrow(ArrowMarker) Add the indicated arrow/line marker to the layer. See class ArrowMarker.

addImage(ImageMarker or string) Add an image marker to the layer. For convenience, you can simply specify the path of the file containing the image in place of an ImageMarker object. In any case, the added ImageMarker is returned.

setTitle(string) Change the layer's title string.

newLegend() Return a new, empty Legend (text marker) object after adding it to this layer.

newLegend(string) Return a new Legend (text marker) object initialized with the given string, after adding it to the layer.

setLegend(string) Set the name of the main legend (i.e., the *real* legend, as opposed to other texts placed on the layer).

legend() Returns the Legend object representing the *real* legend, as opposed to other texts placed on the layer.

removeLegend() Remove the legend (i.e., the object returned by legend()), if one currently exists.

addTimeStamp Add a text marker containing the current date and time.

enableAxis(int,bool) Enable/disable drawing of the indicating axis, which must be one of the integer constants QwtPlot.yLeft (=0), QwtPlot.yRight (=1), QwtPlot.xBottom (=2) or QwtPlot.xTop (=3). The descriptive names were added in SciDAVis 0.2.4; for prior versions, you need to specify the indicated integer values.

setXTitle(string) Set the title displayed for the (bottom) X axis. The axis title can be disabled by setting it to an empty string.

setYTitle(string) Set the title displayed for the (left) Y axis. The axis title can be disabled by setting it to an empty string.

setRightTitle(string) Set the title displayed for the right Y axis. The axis title can be disabled by setting it to an empty string. Added in SciDAVis 0.2.4.

setTopTitle(string) Set the title displayed for the top X axis. The axis title can be disabled by setting it to an empty string. Added in SciDAVis 0.2.4.

setAxisNumericFormat(int, int, int=6, string=QString()) setAxisNumericFormat(axis, format, precision, formula) sets the numeric format used for drawing the axis labels on the specified axis (see enableAxis() for possible arguments). Format must be one of 0 (automatic), 1 (decimal), 2 (scientific) or 3 (superscripts). Precision is the number of digits displayed. If a formula is given, it is evaluated with muParser for every label and the result is used in place of the input value for displaying the label.

setScale(int, double, double, double=0, int=5, int=5, int=0, bool=False) setScale(axis, start, end, step, major_ticks, minor_ticks, type, inverted) sets various options related to the scale of the indicated axis (see enableAxis() for possible arguments); most notably the start and end values of the displayed data range. Step indicates the distance between major tick marks (the default of 0 means to automatically determine a sensible value); alternatively, a target number of major tick marks (and minor tick marks per major tick) can be specified, but due to limitations of Qwt, this is only taken as a hint, not as a strictly followed setting. Type is either 0 (linear scale, default) or 1 (logarithmic scale). The last flag can be set to have the axis inverted, i.e. numbered right to left or top-down.

setMargin(int) Change the margin (i.e., the spacing around the complete content) of the layer.

setFrame(int=1, QColor=QColor(Qt.black)) setFrame(width, color) draws a frame around the complete content (including margin) of the layer; with the indicated line width and color.

setBackgroundColor(QColor) Changes the background color of the layer. Note that this excludes the background of the canvas area containing the curves and markers, which is set separately (see setCanvasColor()). Most commonly, the argument will be either a basic color (Qt.white, Qt.black, Qt.red, Qt.darkRed etc. - see Qt.GlobalColor for details) or a RGBA spec in the form QtGui.QColor(red, green, yellow, alpha) with channels given as integers in the range [0,255] (see QColor for details).

setCanvasColor(QColor) Changes the background color of the canvas area containing the curves and markers. Most commonly, the argument will be either a basic color (Qt.white, Qt.black, Qt.red, Qt.darkRed etc. - see Qt.GlobalColor for details) or a RGBA spec in the form QtGui.QColor(red, green, yellow, alpha) with channels given as integers in the range [0,255] (see QColor for details).

showGrid(int) Toggle drawing of major and minor grid associated with the given axis. See enableAxis() for possible arguments.

showGrid() Toggle drawing of all grid lines.

grid() Returns the Grid object for this layer, which holds various grid-related settings. See class Grid. Added in SciDAVis 0.2.4.

replot() Makes sure that any changes you've applied to the layer are displayed on screen. It is often necessary to call this method after making script-driven changes to a layer, like changing the style of a curve. It is also a good idea to call this before export to a file, although it's not technically required for all file formats.

printDialog() Display dialog for printing out this layer. Added in SciDAVis 0.2.4.

exportImage(string, int=100, bool=False) exportImage(filename, quality, transparent) exports the layer to a bitmap image format with the indicated quality level, optionally making the background transparent (if supported by the file format). The file format is determined by the extension of the indicated file name; the list of supported formats depends on your installation of Qt and can be viewed by invoking the export dialog from the GUI and looking through the "file type" box.

exportVector(string, int=0, bool=True, bool=True, QPrinter.PageSize=QtGui.QPrinter.Custom) exportVector(filename, resolution, color, keep_aspect, page_size) exports the layer to a vector-based file format. You can override the default resolution (in dpi), toggle printing in color/monochrome, toggle whether to keep the width/height aspect when rescaling and select a standard page size as described in the PyQt documentation for QtGui.QPrinter.PageSize.

export(string) Quickly export layer to the indicated file, without bothering about the exportImage()/exportVector() distinction and related options. The file format is determined automatically by looking at the extension of the specified file name; supported formats depend on your installation of Qt and can be viewed by invoking the export dialog from the GUI and looking through the "file type" box.

enableAutoscaling(bool=True) Set automatic updating of the scale settings (see setScale()) when data is added to or changed on the layer.

setIgnoreResize(bool=True) Sets whether to keep the layer's geometry fixed when resizing its graph.

setAutoscaleFonts(bool=True) Sets whether to scale font sizes together with the rest of the layer when its graph is resized (and resizes aren't ignored, see setIgnoreResize()).

setAntialiasing(bool=True, bool=True) setAntialiasing(antialias, update) specifies whether antialiasing should be used for drawing elements added to the layer after the call to setAntialiasing(). If the update flag is True (default), the antialiasing setting of existing elements is updated as well.

setCurveAxes(number, x-axis, y-axis) Note: when using this method the involved axes are autoscaled.See also: setXAxis, setYAxis

canvas() Gives access to the QtGui.QWidget acting as a canvas, i.e it contains all curves and markers on the layer (but not axes and title).

pickPoint() Let the user pick a point on a curve (as with the Data Reader tool) and return the coordinates of the selected point as a QPointF (coordinates can be extracted from the result using QPointF.x() and QPointF.y()). Added in SciDAVis 0.2.4.

4.2.6.13 class Graph (inherits MDIWindow)

A graph window, consisting of layers, which in turn contain plot curves and markers.

```
g = newGraph()
layer1 = g.activeLayer()
layer2 = g.addLayer()
g.setMargins(40,40,40,40)
g.arrangeLayers()
```

activeLayer() Returns the Layer currently marked as active (as indicated by the layer buttons at the top of the graph). If there is only one layer, it is always the active one.

setActiveLayer(Layer) Make the indicated layer the active one.

numLayers() Returns the total number of layers contained in the graph.

layer(int) Returns the Layer specified by index (integer in the range [1,numLayers()]).

layers() Returns a list of all layers in the graph. Added in SciDAVis 0.2.4.

addLayer(int=0,int=0,int=0,int=0) addLayer(x,y,width,height) adds a new layer at the given position and with the given size. Returns the newly created Layer object.

setCols(int) Set the number of columns to use when arranging layers automatically. Doesn't take effect until a re-arrangement is requested by calling arrangeLayers().

setRows(int) Set the number of rows to use when arranging layers automatically. Doesn't take effect until a re-arrangement is requested by calling arrangeLayers().

setSpacing(int,int) setSpacing(row_gap, column_gap) sets the spacing to use between rows and columns when arranging layers automatically. Doesn't take effect until a re-arrangement is requested by calling arrangeLayers().

setMargins(int,int,int,int) setMargins(left,right,top,bottom) sets the outer margins to use when arranging layers automatically. Doesn't take effect until a re-arrangement is requested by calling arrangeLayers().

setLayerCanvasSize(int, int) setLayerCanvasSize(width,height) Resizes all layers as indicated and arranges them automatically.

setAlignment(int, int) setAlignment(horizontal, vertical) sets the alignement of layers in their respective columns and rows. Arguments can be 0 For center, 1 for left/top and 2 for right/bottom. Added in SciDAVis 0.2.4.

arrangeLayers(bool) Automatically arranges the layers on the graph, subject to built-in defaults or settings made previously using setCols(), setRows(), setSpacing(), setMargins() and setAlignment().

exportImage(string, int=100, bool=False) exportImage(filename, quality, transparent) exports the graph to a bitmap image format with the indicated quality level, optionally making the background transparent (if supported by the file format). The file format is determined by the extension of the indicated file name; the list of supported formats depends on your installation of Qt and can be viewed by invoking the export dialog from the GUI and looking through the "file type" box.

exportVector(string, int=0, bool=True, bool=True, QPrinter.PageSize=QtGui.QPrinter.Custom) exportVector(filename, resolution, color, keep_aspect, page_size) exports the graph to a vector-based file format. You can override the default resolution (in dpi), toggle printing in color/monochrome, toggle whether to keep the width/height aspect when rescaling and select a standard page size as described in the PyQt documentation for QtGui.QPrinter.PageSize.

export(string) Quickly export graph to the indicated file, without bothering about the exportImage()/exportVector() distinction and related options. The file format is determined automatically by looking at the extension of the specified file name; supported formats depend on your installation of Qt and can be viewed by invoking the export dialog from the GUI and looking through the "file type" box.

printDialog() Display dialog for printing out this layer. Added in SciDAVis 0.2.4.

4.2.6.14 class Note (inherits MDIWidget)

A note/script window containing arbitrary text, all or parts of which can be executed as Python code. Triggering executing of code in a Note currently doesn't work reliably, but this can essentially be circumvented using exec(str(note.text())).

```
n = newNote()
n.setText("Hello World")
n.exportASCII("output.txt")
```

autoexec() Boolean indicating whether the content of Note is automatically executed when loading the project.

setAutoexec(bool) Sets whether to automatically executed the content of the Note when loading the project.

text() Returns the content of the note as a QString.

setText(string) Sets the content of the note.

exportASCII(string) Writes the content of the note to the indicated file.

importASCII(string) Prepends the content of the note by the content of the indicated file.

4.2.6.15 class ApplicationWindow (inherits QMainWindow)

Manages the current project. The project from which a piece of Python code is executed is accessible via the module attribute `scidavis.app`. However, most of the time, this instance is used implicitly via methods copied to the global namespace by the default scidavisrc.py configuration file.

```
for win in app.windows():
  print win.name()
app.activeFolder().save("subproject.sciprj")
```

table(string) Returns the table with the given name, or None if no such table exists.

newTable() Creates a new table with default settings (as if done using the menu) and returns it.

newTable(string, int=2, int=30) newTable(name, columns, rows) creates a new table with the given name and (optionally) numbers of columns and rows, and returns it. If the specified name is already in use, it is changed automatically to a unique name by appending a number to the requested name.

matrix(string) Returns the matrix with the given name, or None if no such matrix exists.

newMatrix() Creates a new matrix with default settings (as if done using the menu) and returns it.

newMatrix(string, int=32, int=32) newMatrix(name, rows, columns) creates a new matrix with the given name and (optionally) numbers of columns and rows, and returns it. If the specified name is already in use, it is changed automatically to a unique name by appending a number to the requested name.

graph(string) Returns the graph with the given name, or None if no such graph exists.

newGraph() Creates a new graph containing one empty layer (as if done using the menu) and returns it.

newGraph(string) newGraph(name) creates a new graph with the given name containing one empty layer (as if done using the menu) and returns it. If the specified name is already in use, it is changed automatically to a unique name by appending a number to the requsted name. Added in SciDAVis 0.2.4.

note(string) Returns the note window with the given name, or None if no such note exists.

newNote() Creates a new empty note and returns it.

newNote(string) newNote(name) creates a new empty note with the given name and returns it. If the specified name is already in use, it is changed automatically to a unique name by generating a default name like "Notes1". This is inconsistent with newTable(), newMatrix and newGraph() (which generate a unique name by appending a number to the *requested* name) and will probably change in a future release.

plot(Table, string, int=1, int=-1) plot(table, column, style, color) plots the named column of the given table. A new graph is created containing one layer, to which the curve is added. The new graph is returned. Optionally, style and color of the curve can be set, where color is an index in the palette used by SciDAVis for automatically assigning colors to new curves and style is one of the following codes:

0 Line

1 Symbols

2 Line and Symbols

3 Columns

4 Area

5 Pie

6 Vertical drop lines

7 Splines and Symbols

8 Vertical steps

9 Histogram

10 Rows

plot(Table, tuple of strings, int=1, int=-1) plot(table, (column1, column2, ...), style, color) works just like the plot method described above, but plots multiple columns together on the same layer.

importImage(string) Reads an image from the specified file and returns a newly created matrix containing the gray-scale values (brightness) of the image's pixels.

importImage() Ask the user to select an image file and imports it to a newly created matrix as described above. Returns the matrix.

plotContour(Matrix) Makes a contour plot from the given matrix in a newly created graph. Returns the graph.

plotColorMap(Matrix) Makes a contour plot from the given matrix, filling contour areas with colors of the default color map. Returns the newly created graph.

plotGrayScale(Matrix) Makes a gray scale plot from the given matrix in a newly created graph. Returns the graph.

windows() Returns a list of all MDIWindow instances in the project.

results The QTextEdit which holds the results log. Can be used to output custom results from your script, although the recommended way of doing this is to create a Note and set its text to what you want to output.

activeFolder() Returns the folder which is currently being displayed.

setActiveFolder(folder) Given a Folder object, changes the active folder to it. Since MDIWindows are always added to the active folder, this is necessary for script-driven creation of objects in specific sub-folders. Added in SciDAVis 0.2.5.

rootFolder() Returns the root folder of the project.

saveFolder(Folder, string) DEPRECATED. SciDAVis 0.2.4 introduces the new method Folder.save(), which is the recommended way of saving folders to a new project file; unless you need the code to run on previous versions.

renameWindow(MDIWindow, string) DEPRECATED. SciDAVis 0.2.4 introduces the new method MDIWindow.setName(), which is the recommended way of renaming windows; unless you need the code to run on previous versions.

clone(MDIWindow) DEPRECATED. SciDAVis 0.2.4 introduces the new method MDIWindow.clone(), which is the recommended way of cloning windows; unless you need the code to run on previous versions.

setPreferences(Layer) DEPRECATED. Prior to SciDAVis 0.2.4, it was necessary to call this on a layer created using Graph.addLayer() if you wanted the layer to be initialized correctly. This is now done automatically as part of addLayer(); doing so again doesn't hurt, but is not necessary any more.

4.2.6.16 class Fit (inherits QObject)

This is the abstract base class of the various models that can be used for least-squares parameter fitting; in other words, you can only create instances of subclasses of this class, but the methods described here are common to all of these classes (ExponentialFit, SigmoidalFit etc.).

Fit(ApplicationWindow, Layer, string) Constructor, taking the ApplicationWindow instance to which the Fit will belong, the Layer containing data to be fitted (and receiving the result curve) and a name for the fitter. Mainly interesting if you want to implement a custom fitter as a subclass of Fit.

fit() Executes the fit (after setting data and parameters). Must be reimplemented by subclasses.

setYErrorSource(ErrorSource, string="") Specify the standard errors of the Y values (compare setDataFromCurve()). ErrorSource is one of Fit.UnknownErrors (no weighting is done and errors are estimated from the variance of the input values), Fit.AssociatedErrors (the yErr column associated with the source column is used), Fit.PoissonErrors (input values are Poisson distributed, i.e. errors are equal to the square root of the values) or Fit.CustomErrors (the name of the column containing the errors is given as the second parameter).

setDataFromCurve(string, Layer) Use the curve (given by name) on the given layer as the fit data. Returns a boolean indicating success (True) or failure (False).

setDataFromCurve(string, double, double, Layer) DEPRECATED. Use the variant above followed by setInterval() instead.

setInterval(double, double) setInterval(from,to) restricts the X interval of the data points included in the fit to [from,to].

formula() Returns the formula of the fit model (in a format suitable for evaluation with muParser).

numParameters() Returns the number of free parameters in the fit.

setInitialValue(int, double) setInitialValue(index, value) specifies the initial value for the parameter given by index.

setInitialValues(sequence) Set initial values of all parameters at once.

guessInitialValues() Try to determine sensible initial values for the parameters automatically. Can be reimplemented by subclasses if they want to support this feature; currently, only SigmoidalFit and MultiPeakFit can do this trick.

setAlgorithm(Algorithm) Set the fitting algorithm; one of Fit.ScaledLevenbergMarquardt, Fit.UnscaledLevenbergMarquardt or Fit.NelderMeadSimplex.

setTolerance(double) Set the tolerance up to which the result has to be determined. The Fit algorithm is iterated until either the desired tolerance or the maximum number of iterations (see setMaximumIterations()) is reached; in the latter case, it aborts with a failure notice.

setColor(int) Set the color of the generated fit curve from the palette used by SciDAVis for automatically assigning colors to new curves. (Specifying a color by name is DEPRECATED, because its counter-intuitive behaviour was more likely to cause trouble than to make things easier.)

setOutputPrecision(int) Set the number of decimal places to use when formatting numbers for output to the results log or to a graph.

generateFunction(bool, int=100) By default, fit results are pasted into the target graph layer as a function curve. Calling generateFunction(False, num_points) causes the fit results to be written into a hidden table (with num_points data points) and plotted from there. This may be useful if you want to use fit results for further calculations.

setMaximumIterations(int) Sets the maximum number of times to iterate the fitting algorithm before giving up and declaring the fit to have failed.

showLegend() Insert information on the fit into the target graph layer.

legendInfo() Return the information inserted by showLegend() as a string. Can be overridden by subclasses in order to customize it.

scaleErrors(bool) Set flag indicating whether to multiply standard errors of final parameters by χ^2/(degrees of freedom). Defaults to False.

results() Returns the final parameters as a tuple of floats.

errors() Returns standard errors of final parameters as a tuple of floats.

chiSquare() Returns the sum of squares of the residuals from the best-fit line.

rSquare() Returns the coefficient of determination of the best-fit line.

parametersTable(string) Creates a Table with the given name and fills it with the final parameter values. Returns a reference to the Table window.

covarianceMatrix(string) Creates a Matrix with the given name and fills it with the covariance matrix of the final parameter values. Returns a reference to the Matrix window.

4.2.6.17 class Folder (inherits QObject)

Folder(name) Construct a new folder with the given name. You can add it as a sub-folder to an existing one using addChild() (see below).

windows() Returns the list of MDIWindows contained in this folder.

name() Returns the name of this folder (a string).

path() Returns the absolute path of this folder within its project (a string).

folders() Returns the list of sub-folders.

folder(string, bool=True, bool=False) folder(name, case_sensitive, partial_match) returns the first subfolder matching the given name. A subfolders is considered to match if its name equals the name argument; if partial_match is True, it is also considered to match if its name starts with the name argument. Matching can be made case-insensitive by setting the case_sensitive argument to False.

findWindow(string, bool=True, bool=True, bool=False, bool=True) findWindow(str, match_name, match_label, case_sensitive, partial_match) returns the first MDI window whose name (if match_name is True) or label (if match_label is True) matches the given string. A name or label is considered to match if it equals the string argument; if partial_match is True, it is also considered to match if its name starts with the name argument. Matching can be made case-sensitive by setting the case_sensitive argument to True.

table(string, bool=False) table(name, recursive) returns the table of the given name in this folder; if no such table exists and recursive is True, sub-folders are searched as well.

matrix(string, bool=False) matrix(name, recursive) returns the matrix of the given name in this folder; if no such matrix exists and recursive is True, sub-folders are searched as well.

graph(string, bool=False) graph(name, recursive) returns the graph of the given name in this folder; if no such graph exists and recursive is True, sub-folders are searched as well.

note(string, bool=False) note(name, recursive) returns the note of the given name in this folder; if no such note exists and recursive is True, sub-folders are searched as well.

rootFolder() Returns the root folder of the project this folder belongs to.

save(string)> Given a file name, saves the content of this folder as a project file. Added in SciDAVis 0.2.4.

addChild(folder) Adds another folder as a sub-folders to this one. Added in SciDAVis 0.2.5.

4.2.7 The Initialization File

This file allows you to customize the Python environment, import modules and define functions and classes that will be available in all of your projects. The default initialization file shipped with SciDAVis imports Python's standard math functions as well as special functions from SciPy and PyGSL(if available). Also, it creates some handy shortcuts, like **`table("table1")`** for **`sci.app.table("table1")`**.

When activating Python support, SciDAVis searches the following places, executing the first file it can find:

1. ~/.scidavisrc.py[c]
2. /etc/scidavisrc.py[c]
3. ./scidavisrc.py[c]

Files ending in .pyc are compiled versions of the .py source files and therefore load a bit faster. The compiled version will be used if the source file is older or nonexistent. Otherwise, SciDAVis will try to compile the source file (if you've got write permissions for the output file).

4.2.7.1 Recommended approach to per-user configuration

In order to give you full control over the process of setting up the Python environment within SciDAVis, a per-user configuration file (.scidavisrc.py) will supersede any system-wide configuration file. That is, GSL and SciPy functions as well as many SciDAVis-specific functions (like table(), newTable(), plot(), etc.) will be missing, unless you explicitly import them into the global namespace. In order to keep the overview over their customizations and profit from updates to the global configuration files (e.g. with new versions of SciDAVis), most users will want to import the global configuration file from within their custom one. Here's how to do this:

```
import sys
sys.path.append("/etc")
import scidavisrc
# your custom stuff
```

Chapter 5

Command Reference

The active items in the menus depend on the active window in the project. If the active window is a spreadsheet, then all the items linked to table functions are enabled and the others are automatically disabled.

5.1 The File Menu

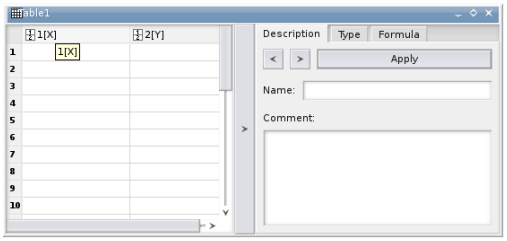
These commands can also be done by clicking on the *New Project* icon from the File toolbar. In order to make this toolbar visible, use the Toolbars command from the View menu.

File→ New →

New→New Project (CTRL+N) Creates a new SciDAVis project file. The project is the main container of SciDAVis, it can include tables, plots and notes. These objects can be organized in folders. If a project is open and saved, it will be closed. If a project is open is not saved, a dialog will be open to ask if the current project has to be saved. The new project will only contain an empty table.

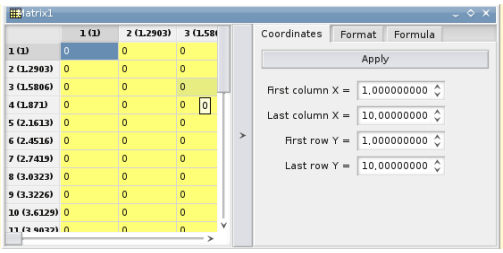
These commands can also be done by clicking on the *New Project* icon from the File toolbar. In order to make this toolbar visible, use the Toolbars command from the View menu.

New→New Table (CTRL+T) Creates a new spreadsheet into the project. This empty table will have 30 rows and 2 columns. This number of rows and columns can be changed with the Dimensions command of the Table menu.



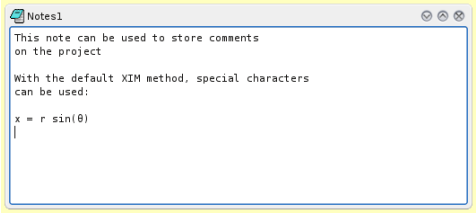
The properties of each column (format of numbers, width, etc) can be modified by the commands of the Table menu. See the table section for more details. There are then many different ways to insert data inside a table: they can be entered one by one, copied and pasted from another software like a spreadsheet, imported from a text file (see Import Ascii command), or filled with the result of a function as explained in the section Filling of a table with the values of a function.

New→New Matrix (CTRL+M) Creates a new Matrix into the project. The empty matrix will have 32x32 cells, these dimensions can be changed by the Dimensions command of the Matrix menu. The default coordinates are ranging between 1 and 10 for x and y.



See the matrix section for more details.

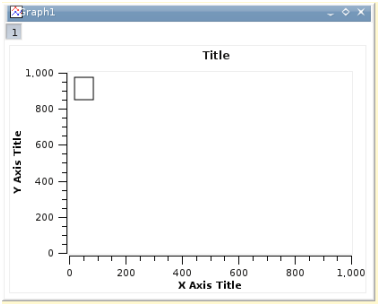
New→New Note/Script (CTRL+ALT+N) Creates a new note window in the project. A note is a simple text window which can be used to add comments to the current project.



This object is also used to store the scripts in python which can be used to perform complex operations with SciDAVis. See the note section and the python scripting sectionfor more details. It can also be used as a calculator.

New→New Graph (CTRL+G) Creates a new empty 2D plot in the project. This default graph is just a framework in which you can add curves from

the columns of a table with the Add/Remove Curve command or define a mathematical expression with the Add Function command (to access to these command, use the Graph menu or do a right click).



The graph will be created with the display parameters selected in the Preferences command (Edit menu).

New→New Function Plot (CTRL+F) Opens a dialog allowing to create a plot by specifying an analytical function. See the 2D plot section of the tutorial for a general overview of this function.

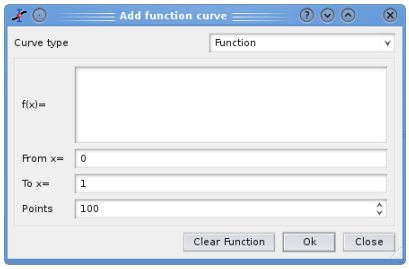


Figure 5.1: The **New→New Function Plot** dialog box.

This function can be defined in cartesian, parametric or polar coordinates, see the Add Function command for more details.

New→New 3D Surface Plot (CTRL+ALT+Z) Opens a dialog allowing to create a 3D plot by specifying an analytical function. See the 3D plot section of the tutorial for more detail on this function. The only available coordinate system is the cartesian one: z=f(x,y).

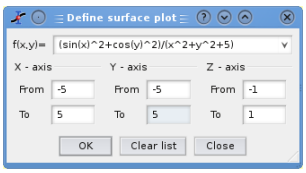


Figure 5.2: The **New→New 3D Surface Plot** dialog box.

You can then enter the X, Y and Z scales.

File → Open (CTRL+O) Opens an existing SciDAVis project file (default file extension *.sciprj*). If your project has been save in a compressed format, you must select the *.sciprj.gz* file format.

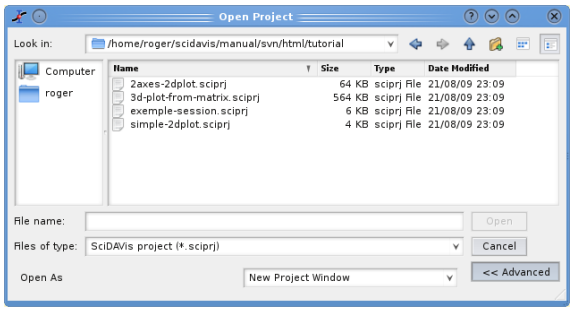


Figure 5.3: The **New→New 3D Surface Plot** dialog box.

This command can also be used to open projects which have been built with the *Qtiplot* software (extension *.qti*) if the version used was below 0.9. By clicking on the *Advanced* button, an additional option appears which allows to insert a project in another as a new folder.

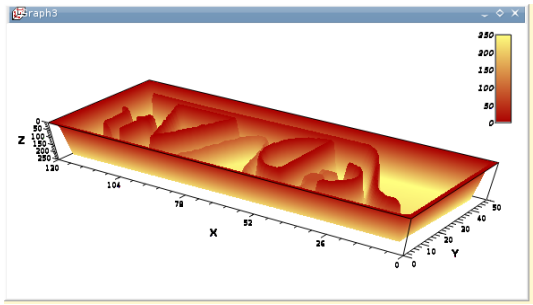
File→ Recent Projects Opens a list of the most recently used SciDAVis project files. You can open one of these files by selecting it from the list. If the file doesn't exist anymore an error message will pop-out and the file will be automatically deleted from the list.

File→ Open image File (CTRL+I) This command loads an image file in a SciDAVis project. This image can be resized and then inserted in another 2D plot. It is in this case similar to the Add Image command. This image can also be used to generate an intensity matrix (see the Import Image command).

Figure 5.4: The result of **Open image File**.

File→ Import Image With this command, an image is loaded in the SciDAVis project

and converted to an intensity matrix. For each pixel, an intensity between 0 and 255 is computed from the intensities of the three colors red, green and blue.



This example shows the 3D plot which has been drawn from the matrix obtained with the SciDAVis logo.

File→ Save Project (CTRL+S) Saves the actual project. If the project hasn't been saved yet ("untitled" project), a dialog will open, allowing to save the project to a specific location.In a project file all settings and all plots are stored in ASCII format.

If the project include large tables, it may be usefull to save the project in a compressed file format. The free zlib library is used to build files in gzip formats (.sciprj.gz).

File→ Save Project As Saves the actual project under a file name different from the current one.

File → Open Template Opens an existing template SciDAVis file. There are four kinds of templates with different extensions for file names.

Entity	Extension	Parameters saved
2D Plot	.qpt	window and layers geometries, fonts and colors for labels and legends, etc. Style for curves is not kept.
3D Plot	.qst	window and layers geometries, fonts and colors for labels and legends, etc
Table	.qtt	number of row and columns
Matrix	.qmt	number of row and columns

You just have to add curves with the Add/Remove Curve command, but the style used to draw the curves is not kept in the template. See the Section 2.1.6.

File → Save as Template Save the active object as a SciDAVis template file. In the case of plot template (.qpt file), the graphical parameters of the plot, together with the text labels (axis, etc) are restored, but the style used to draw the curves and the scales are not saved.

File → Export Graph The plot can be exported into several different image formats. You can define some parameters to customize your image file by checking the *advanced options* button. Depending on the chosen image format, the available options are not the same.

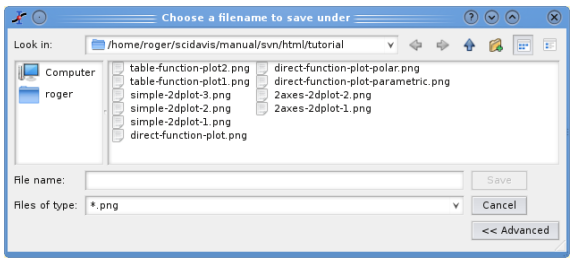
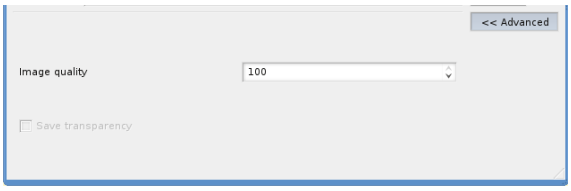


Figure 5.5: The **Export Graph→Current** dialog.

For *tif, bmp, pbm, jpeg, xbm, pgm, ppm* image formats, the quality of the image cannot be controlled, and these formats cannot handle transparency. Therefore, there is no need to check for advanced options.

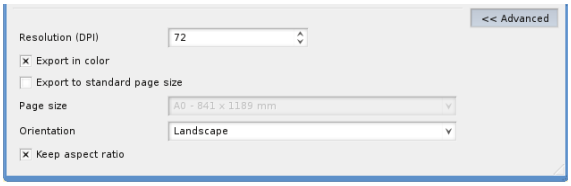
For *jpeg* and *png*, Image Quality parameter ranges between 0 and 100% and defines the compression ratio. The higher it is, the best the quality is but the larger the file is.



For *png*, *tif* and *xpm*, you can choose to use a transparent background.

For *eps, ps* and *pdf* file format, the option dialog is different. The parameters availables are: the size of the paper which is used to draw the plot, and the orientation of the paper sheet. You can choose to keep or not the aspect ratio of the plot, in the last case it will be adapted to the sheet size and orientation.

In addition, you can define the resolution. The default value is 84. If you increase this parameter, the quality of the graphic elements will be better (but the overall size of the plot will be unchanged).



The last format which can be selected is the Scalable Vector Graphic format. With this format, the files can be modified in vector drawing software such as Sodipodi, Inkscape or OpenOffice Draw. You can therefore build more complex images from the pristine SciDAVis plot.

Export Graph→Current (ALT+G) Here you have the possibility to save the active plot under different image formats.

Export Graph→All (ALT+X) Here you have the possibility to save all plots of the project under different image formats. In this case, you must choose a directory for the differents plots. Then one file will be created for each plot, the filename being based on the title of the corresponding window.

File→ Print (CTRL+P) Prints the active plot. A print dialog is opened where you can select the printer, etc.

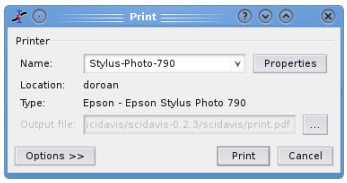
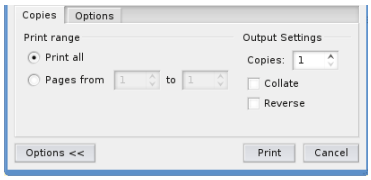
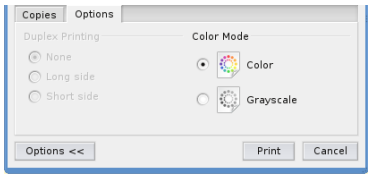


Figure 5.6: The basic **Print** dialog.

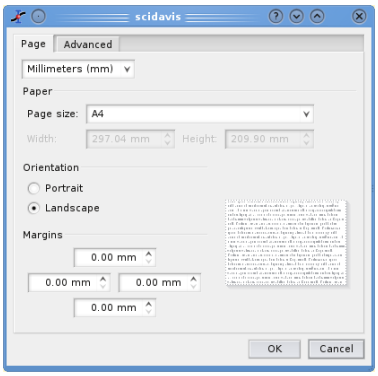
.



If your printer can handle duplex printing and/or color printing, your can select the corresponding options in the *options* tag of this dialog window.



The properties button can be used to select the geometrics parameters of the printed output: paper size, margin, etc.



File→ Print All Plots Prints all plots of the projects. A print dialog is opened where you can select the printer, different paper sizes, etc.

File → Export Ascii Opens a dialog box allowing to save the data from the active spreadsheet to an ASCII file. You can save one selected table, or all the tables of the project. You can then choose the field separator which will be used by SciDAVis. If you check *Export Selection*, only the selected cells will be saved; If not, the whole table will be exported, including the cells with no content.

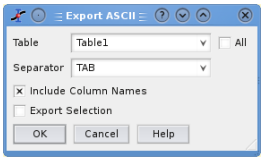


Figure 5.7: The **Export Ascii** dialog.

When the options are selected, click on OK and a new dialog will be displayed to choose the file name. If you check the *all* checkbox, the dialog box will ask for a folder and each table will be save in a file named from the title of the table windows.

File → Import Ascii Imports one or more ASCII file into the project by creating a new spreadsheet storing the data from the file.

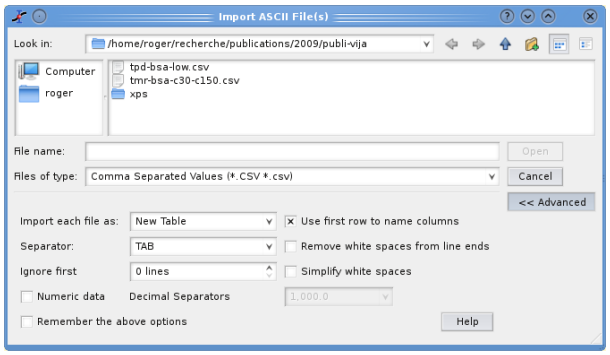


Figure 5.8: The **Import Ascii** dialog.

You can choose to put each data file in a separate table, or join all the data files in one table. There is no automatic analysis of the data. Therefore, by default, the data will be read as text. If you want to obtain directly numeric values, you can specify it in the *numeric data* check box. You must then indicate the format of the numbers. The other possibility is to read data as text and then to specify the type and format of the different columns with the properties dialog of the tables.

If you check the *Remember the above options*, the selected parameters will be used as default values. They will be used if you read an ascii file directly from the command line (see the Command line options section for more details.

File → Quit (CTRL+Q) Closes the application. You will be asked wether you want to save your last changes or not.

5.2 The Edit Menu

Edit → Undo (CTRL+Z) Undo the last command done on tables or matrix. It can also be accessed by clicking on the icon of the edit toolbar. The list of commands which are in the stack can be seen with the Undo/Redo History command.

This command is not available for plot windows.

Edit → Redo (CTRL+R) Restores the modifications in a table after a "Undo" operation. It can also be accessed by clicking on the icon of the edit toolbar. The list of commands which are in the stack can be seen with the Undo/Redo History command.

This function is not available for plot windows.

Edit → Cut Selection (CTRL+X) Copies the current selection to the clipboard and deletes the selection. It can also be accessed by clicking on the icon of the edit toolbar. The command currently works for spreadsheets and for 2D plots objects.

Edit → Copy Selection (CTRL+C) Copies the current selection to the clipboard. It can also be accessed by clicking on the icon of the edit toolbar. The command currently works for spreadsheets and for 2D plots objects.

Edit → Paste Selection (CTRL+V) Pastes the content of the clipboard to the active window. It can also be accessed by clicking on the icon of the edit toolbar. The command currently works for spreadsheets and for 2D plots objects.

Edit → Delete Selection (del) Cleares the current selection. It can also be accessed by clicking on the icon of the edit toolbar. The command currently works for spreadsheets and for 2D plots objects.

Edit → Delete Fit Tables Each time yo do a fit of your data with some mathematical model, a new table is created to put the results of the fit (i.e. the values computed by the model). These tables can be used to plot comparisons of experimental and fitted values.

If you have done several fitting tentatives, a number of unused table may be present in your project. This command allows to remove the results of all the differents fits that you have tested.

Edit → Clear Log Information Deletes from the project file all the history information about the analysis operations performed by the user (fitting, integrations, etc). The log panel is then empty. If your project is reload from a file, all the fitting will be done again and the log-panel will be filled.

Edit → Preferences The preference dialog is used to customize the application. It has five different tabs. If you confirm your changes to the default behaviour of the application, the changes are saved and stored imediatelly.

The first icon can be selected to change the *General* options of the application. In the first tab: *Application*, the style is the general decoration used for the windows. It defines the aspect of the buttons and dialog boxes, as an example all screenshots presented in this manual have been done with the Cleanlooks style available in KDE. The available styles are part of the Qt library. The font is the general font used for the GUI (menus, dialogs, etc), it doesn't apply to the plots. You can select the language of the application in the corresponding combo-box. All the available translations can be downloaded from the following address: Sourceforge repository, you can also use the Translations command from the Help menu.The translated messages are in a file with the extension *.qm* which must be placed in a folder called *share/translations/*, situated in the same location as the SciDAVis executable, in order to be loaded by the application.

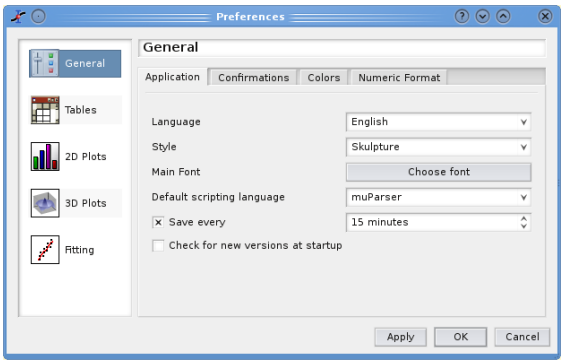
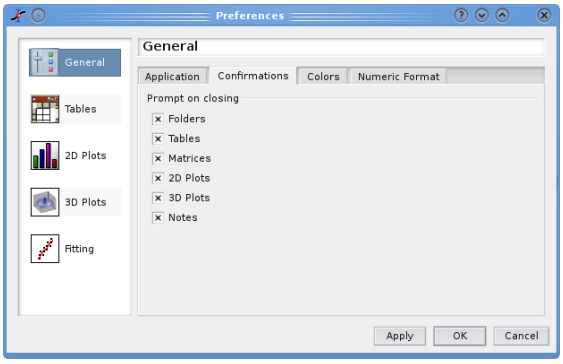
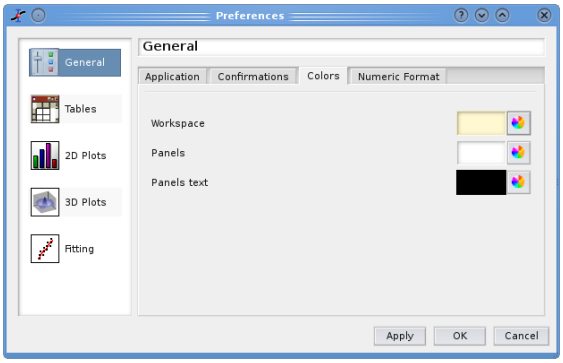


Figure 5.9: The general options dialog: application options.

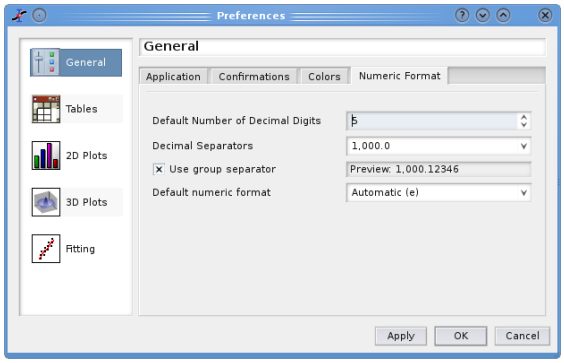
The second tab of the *General* option set is used to disable the prompting on deleting of objects.



In the third tab, you can change the default color for the workspace of the application. You can also choose the background color and the text color for panels. The panels are the Log Window (activated by the Results Log command) and the Project explorer (activated by the Project Explorer command.



The last tab is use to set the default format for numbers. This format will be used in any new table of matrix. If you check the *Update...* option, the decimal separator of the numbers already present in tables will be modified.



The other icons can be used to define the default behaviour of specific objects. Refer to the corresponding sections for more details: tables, 2D links, 3D links and Fitting.

5.3 The View Menu

View → Toolbars There are seven toolbar which can be used to quickly access to the different functions.

File toolbar

edit toolbar

graph toolbar

Plot toolbar

Table toolbar

Matrix-plot toolbar

3D surface toolbar

In most cases, they are present automatically when necessary.

View → Plot Wizard (CTRL+ALT+W) This dialog is used to build a new plot by selecting the columns in the tables available in the current project. At first, you have to select the table you want to use, and then click on *New curve* to create the curve. After that, you have to select at least one column for X and one for Y. You can also select one more column for X-errors or for Y-errors. The plot created will have the default style you defined using the 2D plot preferences dialog through the '2D Plots -> Curves' tab.

In this example, one curve is selected from the first table, and the other from the second table (with X error bars)

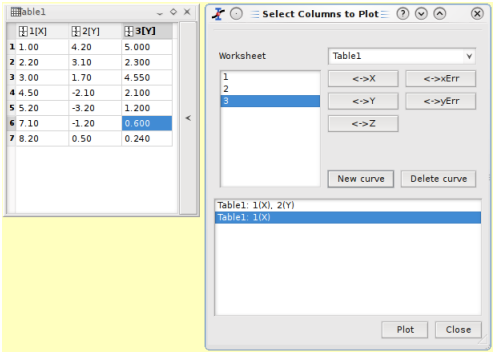


Figure 5.10: The plot wizard dialog box.

View → Project Explorer (CTRL+E) Opens/Close the Project Explorer, which gives an overview of the structure of a project and allows the user to perform various operations on the windows (tables and plots) in the workspace.

The project explorer shows a list of all the windows, tables, matrices and folders which are included in the current project. It can be used to create new folders and windows, to find existing ones, to make hidden elements visible, to perform basic operations like: renaming, deleting, hiding, resizing, printing, etc... You can also use it in order to display the list of dependencies and properties of an element in the project.

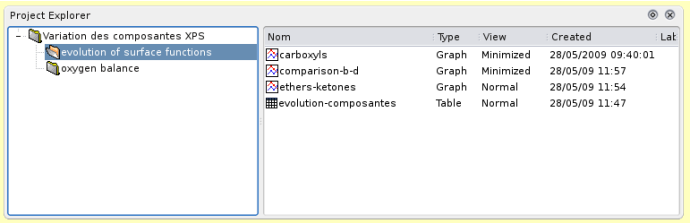


Figure 5.11: The project explorer panel.

View → Results Log Opens/Close a panel displaying the historic of the data analysis operations performed by the user.

View → Undo/Redo History This command shows a window which contain all the command which have been done on tables and matrix during the session.

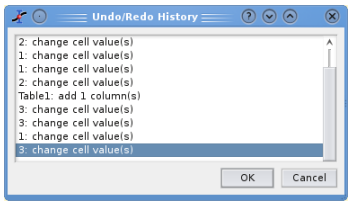


Figure 5.12: The undo-redo history.

View → **Console** .

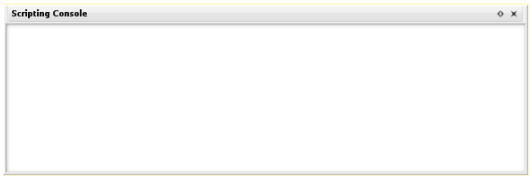


Figure 5.13: The scripting console.

5.4 The Graph Menu

This menu is only active when a plot window is selected.

Graph → **Add/Remove Curve (ALT+C)** Opens the **Add/Remove Curve** dialog, allowing to easily add or remove curves from the active plot layer. This dialog can also be used to modify a curve which is already plotted by changing the columns which are used as X or Y values.

The left window shows the columns which are available for plotting in the different tables of the project, and the right window gives the list of the curves already plotted. In the case presented below, there are two tables in which the **Add/Remove Curve** dialog box allows to select columns. If you use this dialog box to add a column, the X column will be the one define as X in the corresponding table.

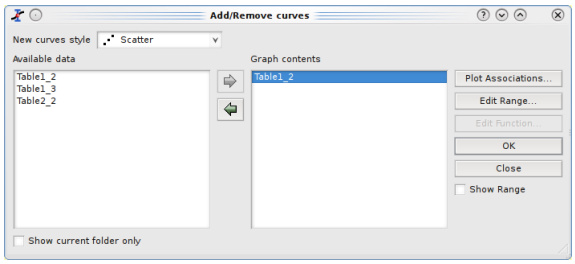


Figure 5.14: The **Add/Remove Curve** dialog box.

In this dialog box, if you select one curve of the plot in the right window, you can change the columns used for X and Y with the *Plot Association* button. In any case, you can't mix the X values of one table with the Y values of another one. If you wan't to do this, you have to copy the columns in the same table.

If the curve selected is a function, you can modify it. Refer to the Add Function command for more details on functions editing.

Graph → Add Error Bars (CTRL+B) This command is used to plot X and/or Y error bars around the data points.

It must be taken care that the "add" button add the errors bars, and so do the "OK" button. Then, you should close the dialog with cancel if you have clicked on the "add" button.

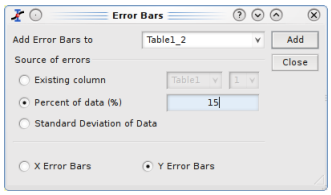


Figure 5.15: The **Add Error Bars** dialog.

There are three ways to specify the size of the bar:

A column of the table In this case, the values of the selected column are used to compute the error bars. if V is the value of the data point, and E the value of the errorbar column, the size of the bars will be V-E to V+E.

A percentage of the values if E is the percentage selected, the size of the bars will be V(1-E/100) to V(1+E/100). It must be noticed that, in addition to the errorbars on the plot, this command will create a new column in the active table with can be used in the way as with the previous option. This column can be modified like any other one.

The standard deviation of the values the standard deviation of the values. This has a meaning only of the data are centered around an average value. Like with the previous option, a new column will be created in the active table.

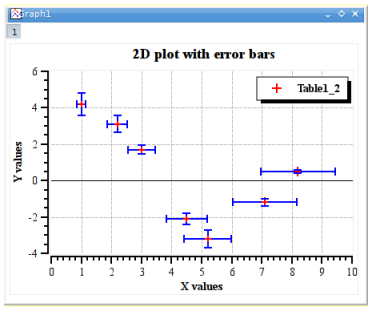


Figure 5.16: A plot with X and Y Error Bars.

This dialog box is used to add a function curve to the active plot. The function can be built with the common operators: * + / - and ˆ for the power. The intrinsic functions available are listed in the appendix.

The most common way to define a function is the classical cartesian coordinate definition y=f(x), this is the defaut option. The two following parameters allow to select the x range used for the plot, and the last one is used for the number of data points that are computed in the X-range.

Graph → Add Function (CTRL+ALT+F)

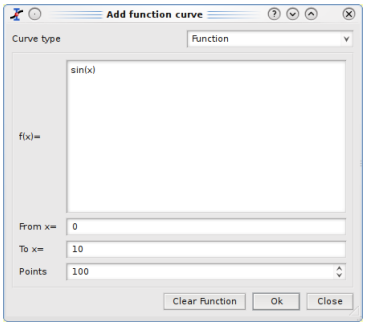


Figure 5.17: The **Add Function** dialog box: cartesian coordinates.

The functions can also be defined in a parametric definition: if *t* is the parameter, the (x,y) data points are computed by x=f(t) and y=g(t). The first parameter is the name of the parametric variable (here *t*) followed by the range, the definition of the two functions and the number of data points.

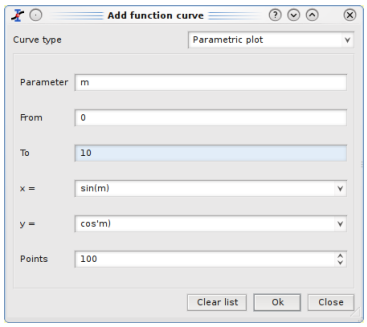


Figure 5.18: The **Add Function** dialog box: parametric coordinates.

The last way is the polar definition of the function: if *t* is the parameter, the radius *r* and the angle *theta* are computed by r=f(t) and theta=g(t). Then the (x,y) data points are computed by x=r*cos(theta) and y=r*sin(theta).

The first parameter is the name of the parametric variable (here *t*) followed by the range, the definition of the two functions and the number of data points.The angle is defined in radians, and the constant value *pi* can be used: it is possible to use 3*pi to define the parameter range.

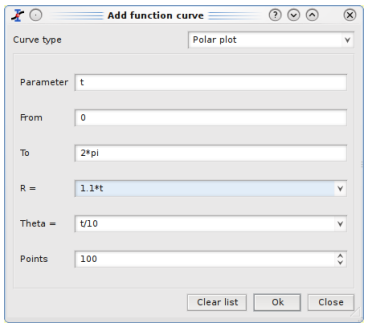


Figure 5.19: The **Add Function** dialog box: polar coordinates.

Graph → Add Text (ALT+T) Opens a dialog allowing you to select whether the text is to be added to the active plot layer or on a new layer. The cursor changes to an edit text cursor. Next, you must click in the plot window to specify the position of the new text box. A text dialog will pop-up allowing you to type the new text to be displayed and all its properties (color, font, etc...)

If you choose the *On new layer* option, the text will be inserted as a new layer which has the size and the position of the text. You can then modify the size and position of this layer with the *layer Geometry* (see the Add Layer command for details). Beware that in this case, all text which is not in the layer will be clipped,

therefore, you need to modify the layer to modify the position of the text. If you choose the *On Active layer* option, the text will be inserted in the selected layer, and its position can be modified directly with the mouse inside this layer.

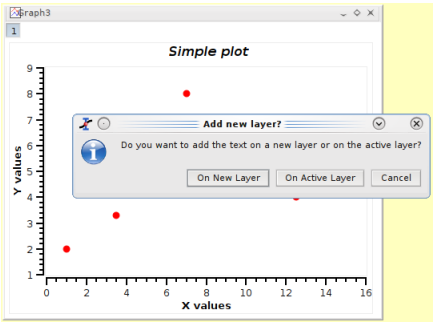


Figure 5.20: The **Add Text** dialog box.

Graph → Draw Arrow (CTRL+ALT+A) Changes the active layer operation mode to the drawing mode. You must click on the layer canvas in order to specify the starting point for the new arrow, and then click once more to specify its ending point. You can edit the new arrow using the Arrow dialog. You can swith back to the normal operating mode by clicking the "Pointer" icon in the Plot toolbar.

Then, a dialog allows to modify a line or an arrow which has been created. One can open it with a double click on an arrow or a line, or by selecting an arrow or a line and selecting *Properties...* with the right button of the mouse.

The first tab allows to change the color, the line type and the line width. This last parameter is set in pixels. It is possible to define a default style for all the new lines by pressing the *Set Default* button.

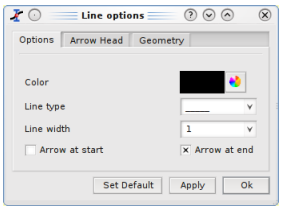


Figure 5.21: The *Arrow options* dialog: first tab

The *Arrow head* tab is used to modify the shape of the head of the arrow. The length is set in pixels and the angle is in degrees. It is also possible to define a default style for the arrow heads using the same *Set Default* button.

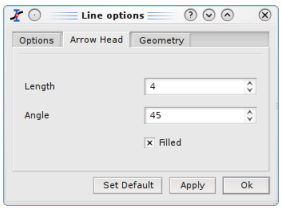


Figure 5.22: The *Arrow options* dialog: second tab

The *Geometry* tab allows to specify the start and end points of the line/arrow. The coordinates can be set as a function of the scales values displayed on the left axis (Y) and on the bottom axis (X) or in pixels, by choosing the desired method from the *Unit* pull-down list. The pixel coordinates are relative to the top-left corner of the layer which contains the line.

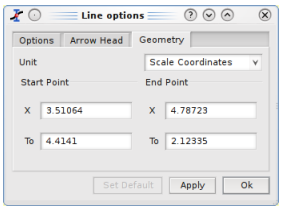


Figure 5.23: The *Geometry* dialog: third tab

Graph → Draw Line (CTRL+ALT+L) Changes the active layer operation mode to the drawing mode. You must click on the layer canvas in order to specify the starting point for the new arrow, and then click once more to specify its ending point. You can edit the new arrow using the line dialog. You can swith back to the normal operating mode by clicking the "Pointer" icon in the Plot toolbar.

Graph → Add Time Stamp (CTRL+ALT+T) This command is used to add a special label in the current plot which contains the current date and time. The properties of this label can be customized like any other label that is added by the Add Text command.

A timestamp label is not modified if the plot is modified, saved, etc.

Graph → Add Image (ALT+I) Opens a file dialog allowing you to select an image to be added to the active plot layer. Only a link to the image file will be saved into the project file and not the image itself. The new image is added to the left-top corner of the layer and can be moved by drag-and-drop.

Graph → New Legend (CTRL+L) Adds a new legend object to the active plot layer. You can have more than one legend on a plot. These legends can then be customized by double clicking on a given legend.

Graph → Automatic Layout Restore the drawind parameters of the layout to its default values (as they are defined in the dialog box of the Arrange Layers command): margins between the layer and the window border of 5 pixels, layer centered in the window, etc.

Graph → Add Layer (ALT+L) This dialog is opened when you want to add a new layer on the active plot. If you select *Guess*, SciDAVis will divide the window in two columns and put the new layer on the right. If you choose *Top-Left Corner*, SciDAVis will create a new layer with the maximum possible size over the existing layer, this layer contains an empty plot.

Figure 5.24: The **Add Layer** dialog box.

You can then modify the size and position of each layer by selecting it with the layer number buttons 1 2 and selecting the *Layer Geometry* command from the context menu.

Graph → Remove Layer (ALT+R) Deletes the active layer and prompts out a question dialog allowing you to choose whether the remaining layers should be automatically re-arranged or not.

Graph → Arrange Layers (ALT+A) This dialog allows to modify the geometrical arrangement of the plots which are already present in the active window. You can also add new layers or remove existing ones.

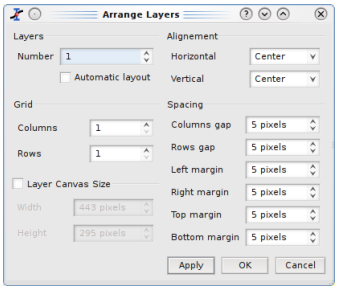


Figure 5.25: The **Arrange Layers** dialog

The *Arrange Layers* dialog is used to modify the geometrical arrangement of the plots. You can specify the numbers of rows and columns which will define a table of plots. As pointed out above, you can also add or remove layers with this dialog, using the "Number of Layers" box.

With the default setting, SciDAVis computes the size of the layers from the size of the window. If you check the *Layer Canvas Size*, you can set the size of the layers and SciDAVis will modify the size of the window.

The two right zones allow to set the alignement of the layers in the window, and the margins between the layer borders and the window limits.

If you do some modifications on your plot, the alignment of the different axis may not be conserved. You can exec again the **Arrange Layers** to re-arrange your plot.

5.5 The Plot Menu

This menu is active only when a table is selected. It can also be accessed in the context menu when one or more columns of a table are selected. These commands allow to plot the data selected in the active table. There are several possibilities to plot from a table:

Conventional X-Y plots: lines, scatter

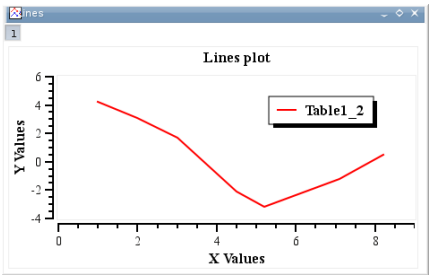
Other plots which are drawn as X-Y plots: columns, rows

Plots which need the computation of a distribution of values from the columns of data: histograms, box plots

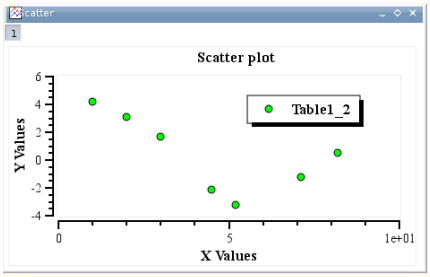
Vector plots which need four columns

3D plots drawn from a set of (X,Y,Z) triplets in three columns

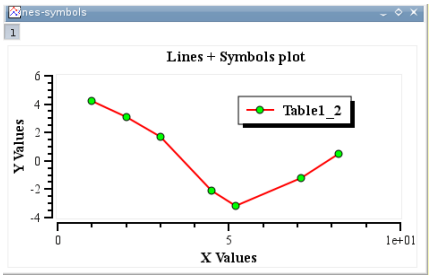
Line Plots the selected data columns in the active table window using the "Line" style. This command can also be activated by clinking on the icon of the Table toolbar. Once the plot is created, the drawing of the data series can be customized (see Section 2.1.4).



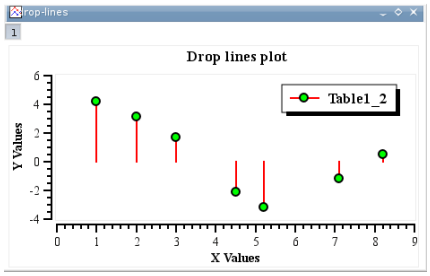
Scatter Plots the selected data columns in the active table window using the "Scatter" style. This command can also be activated by clinking on the icon of the Table toolbar. Once the plot is created, the drawing of the data series can be customized (see Section 2.1.4).



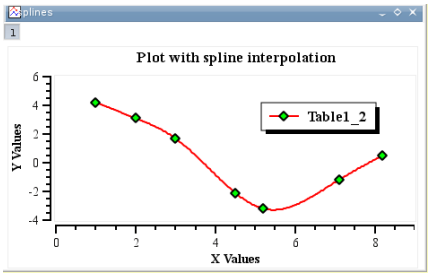
Line+Symbol Plots the selected data columns in the active table window using the "Line + Symbol" style.This command can also be activated by clinking on the icon of the Table toolbar. Once the plot is created, the drawing of the data series can be customized (see Section 2.1.4).



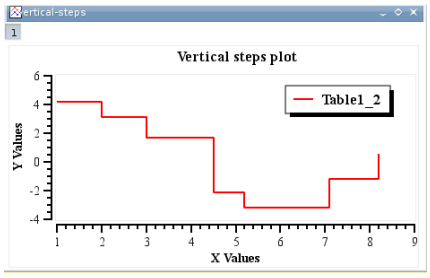
Special Line/Symbol → Special Line/Symbol→Vertical Drop Lines Plots the selected data columns in the active table window using the "Vertical drop lines" style. Once the plot is created, the drawing of the data series can be customized (see Section 2.1.4).



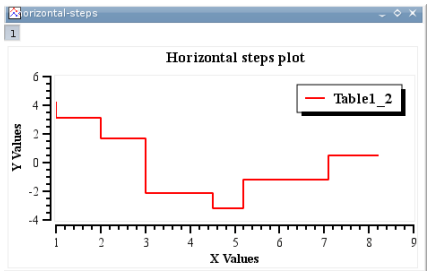
Special Line/Symbol→Splines Plots the selected data columns in the active table window using the "Spline" style. Once the plot is created, the drawing of the data series can be customized (see Section 2.1.4).



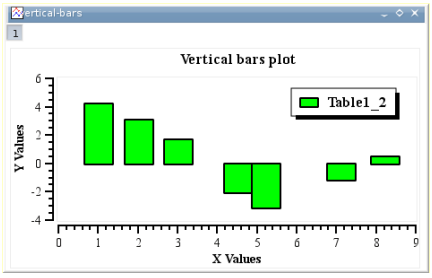
Special Line/Symbol→Vertical Steps Plots the selected data columns in the active table window using the "Vertical Steps" style. Once the plot is created, the drawing of the data series can be customized (see Section 2.1.4).



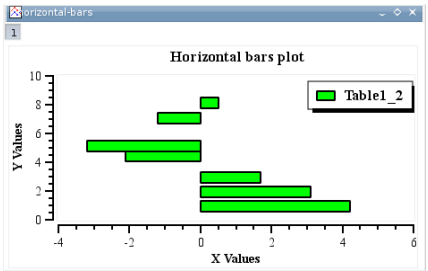
Special Line/Symbol→Horizontal Steps Plots the selected data columns in the active table window using the "Horizontal Steps" style. Once the plot is created, the drawing of the data series can be customized (see Section 2.1.4).



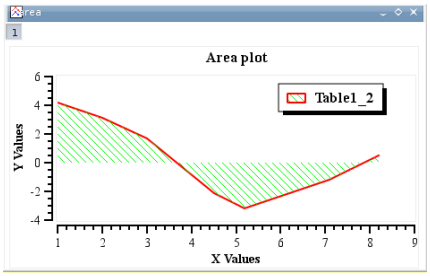
Vertical Bars Plots the selected data columns in the active table window using the "Columns" style, that is vertical bars.



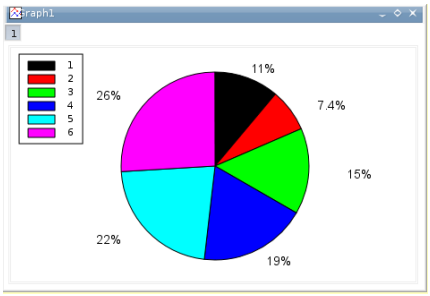
Horizontal Bars Plots the selected data columns in the active table window using the "Rows" style.



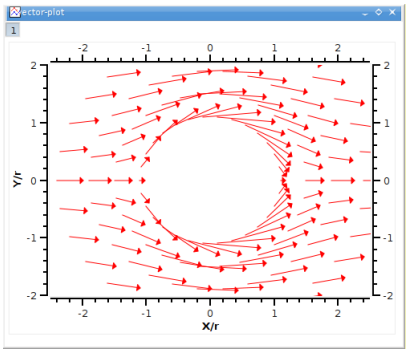
Area Plots the selected data columns in the active table window using the "Area" style, that is a line style with the area under the curve filled.



Pie Creates a 2D Pie plot of the selected column in the active table window (only one column allowed). See Section 2.2.1 for more details.



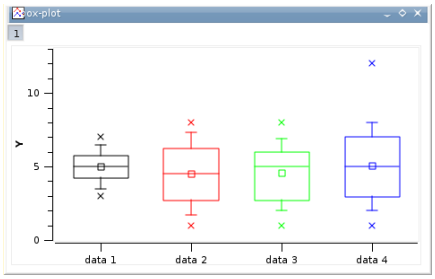
Vectors XYXY Creates a vectors plot of the selected column in the active table window. You must select four columns for this particular type of plot. The two first columns give the coordinates for the starting points of the vectors, the two last columns giving the information regarding the end points. See Section 2.2.2 for more details.



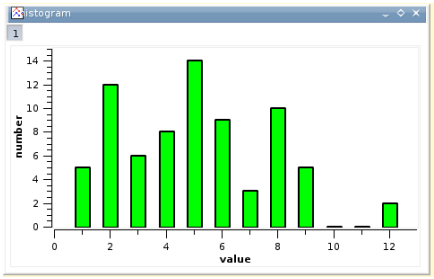
Vectors XYAM Creates a vectors plot of the selected column in the active table window. You must select four columns for this particular type of plot. The two first columns give the coordinates for the starting points of the vectors, the two last columns giving the angle (in radians) and the magnitude of the vectors. See Section 2.2.2 for more details.

Statistical graphs Statistical plot will not give a direct drawing of the data selected in the table, but they will give a representation of the frequency distribution of the Y-values.

Statistical graphs→Box Plot Creates a box plot of the selected data columns in the active table window. This type of plot is used to give a graphical representation of the some classical parameters of the frequency distribution such as the mean of data, the min and max values, the position of the 95 and 5 percentiles, etc. The choice of the statistical parameters and the graphical parameters can be modified (see Section 2.3.1).



Statistical graphs→Histogram Creates a frequency histograms of the selected data columns in the active table window.



With this command, a frequency distribution is computed from your data. The default binning uses 10 steps between the max and the min of Y-values. The parameters used to compute the distribution and the graphical parameters used for the drawing of the columns can be modified (see Section 2.3.2 for details).

If you want to draw an histogram directly from values, use the 3D Plots→Bars command.

Statistical graphs→Stacked Histogram Creates vertically stacked layers displaying the histograms of the selected data columns in the active table window (one histogram per layer) See the Panel→Vertical 2 Layers command for more details.

Panel These commands can be used to obtain quickly some classical arrangements of multiple plot.

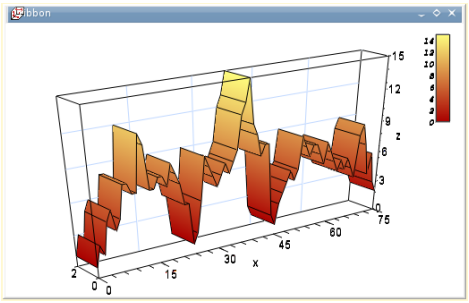
Panel→Vertical 2 Layers Creates 2 vertically stacked layers displaying the selected data columns in the active table window (one curve per layer).

Panel→Horizontal 2 Layers Creates 2 horizontally stacked layers displaying the selected data columns in the active table window (one curve per layer).

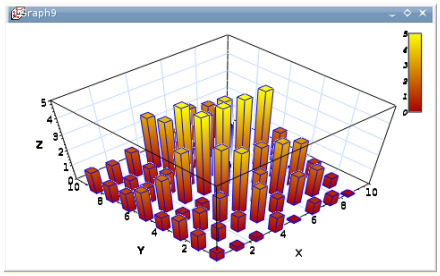
Panel→4 Layers Creates 4 layers on a 2x2 grid, displaying the selected data columns in the active table window (one curve per layer).

Panel→Stacked Layers Creates vertically stacked layers displaying the selected data columns in the active table window (one curve per layer).

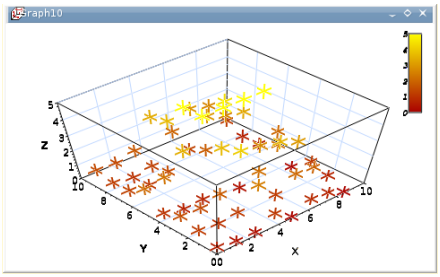
3D Plots **3D Plots→Ribbon** Makes a 3D plot of the selected data column in the active table window (only one column allowed) using the "Ribbon" style.



3D Plots→Bars Makes a 3D plot of the selected data column in the active table window (only one column allowed) using the "3D Bars" style.

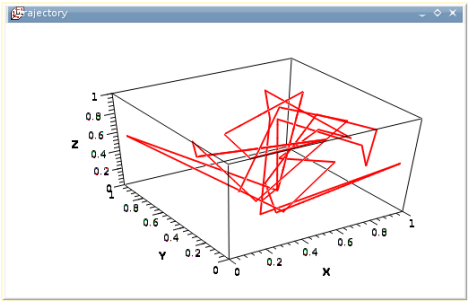


Scatter Makes a 3D plot of the selected data column in the active table window (only one column allowed) using the "3D Dots" style. The 3D point symbol style can be changed via the 3D Plots Settings dialog.



With scatter plots, you can choose the kind of graphic item which is used to plot the data points. The example above is done with cross hairs, but you can also select points or cones. This can be done either with the corresponding icons of the 3D surface toolbar (respectively and for cross-hairs, dots and cones) or with the custom-curves dialog.

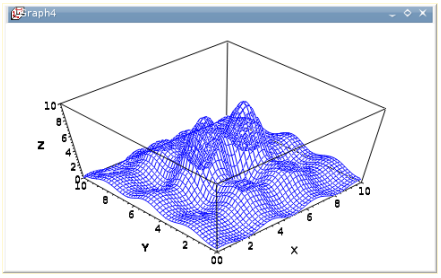
3D Plots→Trajectory Makes a 3D plot of the selected data column in the active table window (only one column allowed) using the "3D Line" style. The line width and color may be changed via the 3D Plots Settings dialog.



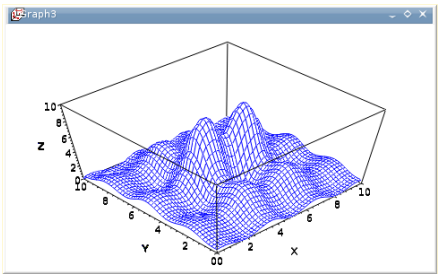
5.6 The Plot 3D menu

This menu is only active when a matrix is selected.

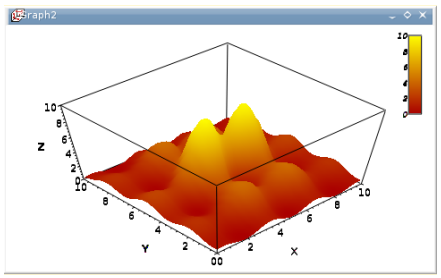
3D Wire Frame Makes a 3D plot of the selected matrix using the "3D mesh" style.



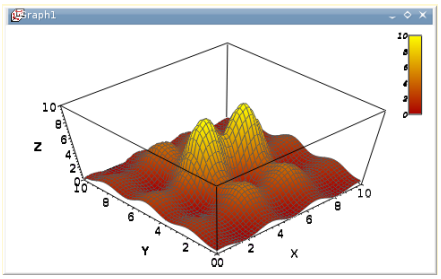
3D Hidden Lines Makes a 3D plot of the matrix using the "3D mesh" style with hidden lines.



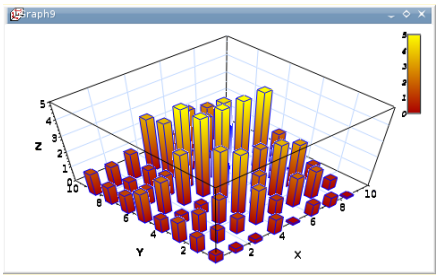
3D Polygon Makes a 3D plot of the matrix using the "3D polygons" style.



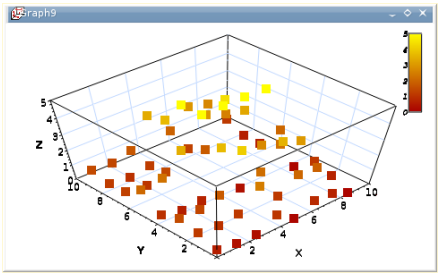
3D Wire Surface Makes a 3D plot of the matrix using the "3D polygons" style with the mesh drawn.



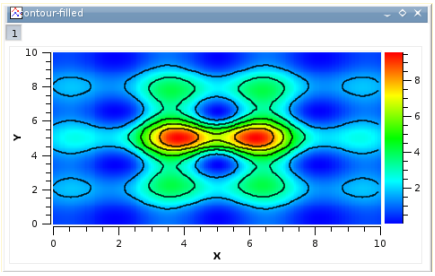
3D Plots→Bars Makes a 3D plot of the selected data column in the active table window (only one column allowed) using the "3D Bars" style.



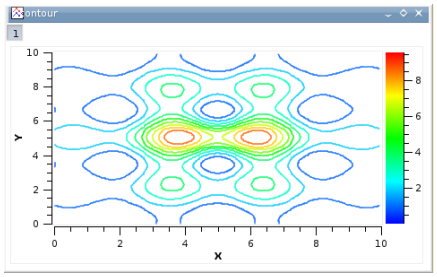
Scatter Makes a 3D plot of the selected data column in the active table window (only one column allowed) using the "3D Dots" style. The 3D point symbol style can be changed via the 3D Plots Settings dialog.



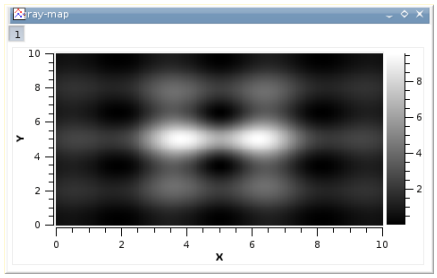
Contour+Color Fill Makes a color map plot of the data in the active matrix window. The contour lines and the colormap settings may be changed by clicking on the plotting area, this will active the *Contour Options Dialog*.



Contour Lines Makes a contour plot of the data in the active matrix window. The contour lines and the colormap settings may be changed by clicking on the plotting area, this will active the *Contour Options Dialog*.



Gray Scale Map Makes a gray map plot of the data in the active matrix window. The contour lines and the colormap settings may be changed by clicking on the plotting area, this will active the *Contour Options Dialog*.



5.7 The Tools Menu

This menu is active only when a plot is selected. Its commands can also be accessed by clicking on the icons of the graph toolbar

Data -> Disable Tools When you are using a command which modify the pointer such as the **Data Reader**, this command can be used to exit this special mode, and go back to the normal pointer behaviour.

Data -> Zoom In (CTRL++) Switches the active plot layer to the zoom mode. The mouse cursor shape changes to a magnifying lens only inside the active plot canvas. You can select a window in the current plot which will be used as the new plotting window.

Data -> Zoom Out (CTRL+-) This command cancel the previous zooming, a history of the zoom is kept so that you can do multiple zoom out commands.

Data -> Rescale to Show All (CTRL+SHIFT+R) Rescale the active plot layer to its default parameters, and therefore cancel all the zoom operations which have been done.

Data -> Screen Reader Opens the Data Display toolbar and changes the mouse cursor shape to a small cross target. By keeping the left button pressed and moving the mouse you can view the coordinates of the cursor with respect to the axes of the active plot layer.

Data -> Data Reader (CTRL+D) Shows a red cross cursor and opens the Data Display toolbar giving easy and fast access to the values of the data points. You can select data points by moving the cursor with the Left and Right arrow keys or faster by clicking on them with the mouse. You can navigate through the curves on the plot layer using the Up and Down arrow keys.

Data -> Select Data Range (ALT+S) Shows two rectangular cursors that can be used for selecting the data range when performing analysis operations. The mouse cursor shape changes to a rectangular target only inside the active plot canvas. The active cursor is red, the other is black.You can move the active cursor with the arrows keys while keeping the Ctrl key pressed or faster by clicking on a curve point. You can change the active cursor using the Left and Right arrow keys. You can navigate through the curves on the plot layer using the Up and Down arrow keys.

Data -> Move Data Points (CTRL+ALT+M) Allows you to modify the position of data points in the active plot layer by simple drag-and-drop. It opens the Data Display toolbar, for a better visualisation of the new coordinates.

The changes you make automatically modify the data into the corresponding tables and all the plots depending on those data sets. You can cancel the modifications with the Undo command.

Data -> Remove Bad Data Points (ALT+B) Allows you to remove data points from the active plot layer by double-clicking on them. The coordinates of the points selected for removal are shown in the Data Display toolbar.

The changes you make automatically modify the data into the corresponding tables and all the plots depending on those data sets. You can cancel the modifications with the Undo command, but you need to undo twice ro restore a point: the first one to create the removed point, and the second to put it at the right place in the plot.

5.8 The Analysis Menu

The commands which are available in this menu are not the same if a table or a plot is selected. For most analysis commands, you car refer to the tutorial in Chapter 3.

5.8.1 Commands for the analysis of data in tables

Statistics on Columns Creates a new table providing basic statistical information about the selected columns in the active table: average, variance, standard deviation, max value, etc...

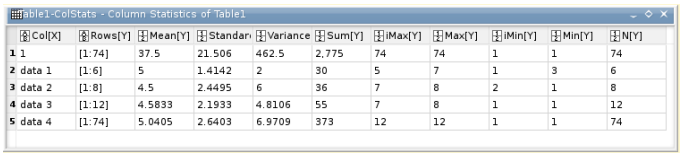


Table1-ColStats - Column Statistics of Table1

	Col[X]	Rows[Y]	Mean[Y]	Standard	Variance	Sum[Y]	iMax[Y]	Max[Y]	iMin[Y]	Min[Y]	N[Y]
1	1	[1:74]	37.5	21.506	462.5	2,775	74	74	1	1	74
2	data 1	[1:6]	5	1.4142	2	30	5	7	1	3	6
3	data 2	[1:8]	4.5	2.4495	6	36	7	8	2	1	8
4	data 3	[1:12]	4.5833	2.1933	4.8106	55	7	8	1	1	12
5	data 4	[1:74]	5.0405	2.6403	6.9709	373	12	12	1	1	74

You can select several columns in one table, one line will be created for each column. You can't select columns in different tables to obtain one single table of statistics.

Statistics on Rows Creates a new table providing basic statistical information about the selected rows in the active table: average, variance, standard deviation, max value, etc...

See the Statistics on Columns command command for more details.

FFT Computes a direct or inverse Fast Fourier Transform. See the Section 3.1 of the Chapter 3 for more details.

Correlate Does a cross-correlation of the two columns which are selected. See the Section 3.3 of the Chapter 3 for more details.

Correlate Does a correlation of the selected column with itself. See the Section 3.3 of the Chapter 3 for more details.

Convolute Does a convolution of the two columns which are selected. The first one being the response and the second the signal. See the Section 3.4 of the Chapter 3 for more details.

Deconvolute Does a deconvolution of the two columns which are selected. The first one being the response and the second the signal. See the Section 3.5 of the Chapter 3 for more details.

Fit Wizard (CTRL-Y) Opens the *Non-linear Fit* dialog, allowing you to choose the curve to fit, the algorithm and the tolerance, the number of iterations to be performed, and to type the analytical function to use, the names of the fitting parameters and their initial guessed values. See the Section 3.6.1 of the Chapter 3 for more details.

5.8.2 Commands for the analysis of curves in plots

The following items are enabled only if the active window is a 2D Multilayer Plot Window. If the active plot layer contains more than one curve, and the Data Range Selectors are not enabled, a dialog window will pop-out allowing you to select the curve you want to analyse.

In most of the cases (except for integration), a new red curve is added to the active plot layer and a a new table containing the data used to plot this curve is added to the workspace. Useful information about the operation performed will be showed in the **Results Log**.

The commands FFT command and Fit Wizard command are presented in the Analysis-tables menu.

Differentiate Creates a new plot displaying the resulting curve of the numerical differentiation. The computation of the derivative is done by centered finite differences over the point before and the point after each data point:

This command creates a new table which contains one column for X-values and one column for derivatives of Y-values. It also creates a new plot of the derivative. The numeric differentiation can generate a lot of noise for a given curve, and a smoothing may be necessary before this operation (see Smooth command).

Integrate Opens the integration dialog, allowing to choose the curve to integrate and the integration method. This command can't be used to obtain a cumulative curve from a selected curve, it can only compute the integral of the data between two limits.

The first field is the curve that will be integrated. The second one is the order of the integration: the order 1 corresponds to the trapezoid rule, i.e. the curve is aproximated by a straight line between 2 successive points. If you choose the order 2, three successive points are used and a second order polynome is used to approximate the curve. etc. If you have a large amount of points in your curve, the order 1 is enough.

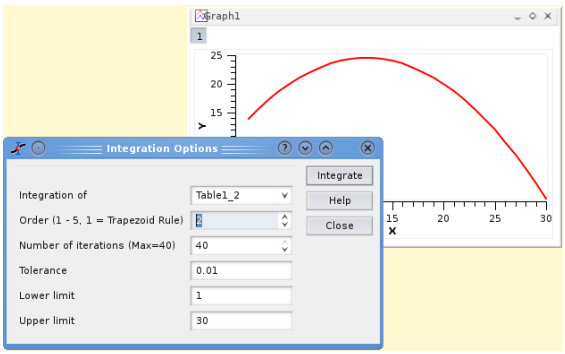
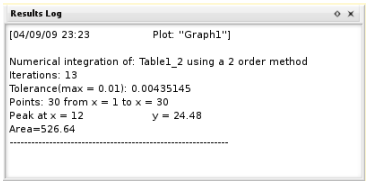


Figure 5.26: The **Integrate** dialog box.

The result of the integration will be given in the **Results Log**.



Smooth These commands will generate a new curve by dooing a smoothing ofthe selected curve.

Smooth→Savitsky-Golay This command performs a smoothing of the selected curve with the Savitzky-Golay method. The formula used to smooth the curve defined by the points $y_i=f(x_i)$ is:

$$z_i = \frac{1}{n} \sum_{j=i-n/2}^{j+n/2} f_i y_i$$

The f_i values are computed by fitting the data points to a polynome, they depend on the number of points used for the smoothing of the curve and the order of the polynome. Compared to the moving window average method, the advantage of this smoothing method is that the values of extrema are not truncated. The dialog allows to specify the curve which will be smoothed, the value of the order of the polynome, the number of data points used for the polynomial fit before and after each point and the color used to draw the smoothed curved. A new table will be created to store the data points x_i, z_i.

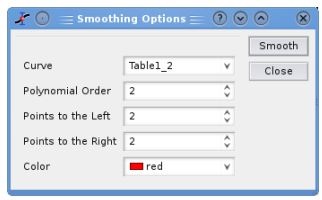


Figure 5.27: The **Smooth→Savitsky-Golay** dialog.

Smooth→Moving Window Average This command performs a smoothing of the selected curve with the moving window average method. The formula used to smooth the curve defined by the points $y_i=f(x_i)$ is:

$$z_i = \frac{1}{n} \sum_{j=i-n/2}^{j+n/2} y_i$$

The greater the number of points *n*, the smoother the resulting curve $z_i=f(x_i)$ is. The dialog allows to specify the curve which will be smoothed, the value of *n* and the color used to draw the smoothed curve. A new table will be created to store the data points x_i, z_i.

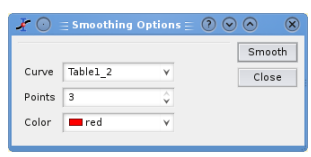


Figure 5.28: The **Smooth→Moving Window Average** dialog.

Depending on the number of data points and on the variation of the Y values, smoothing can give very different results.

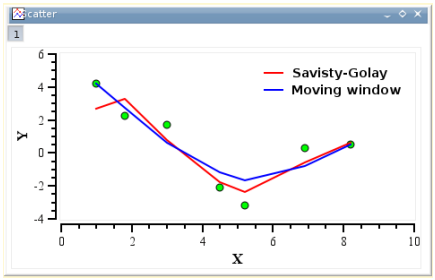


Figure 5.29: Comparison of the two smoothing methods.

Smooth→Moving Window Average This command allow a smoothing based on FFT filtering of data. It can be used when you have noisy curves with a large number of data.

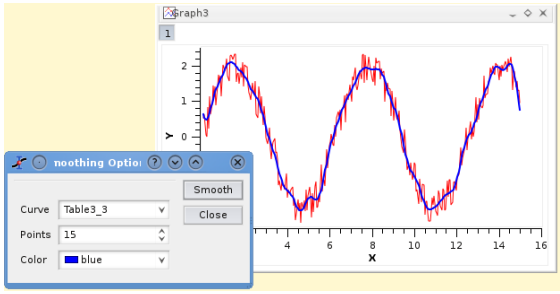


Figure 5.30: The dialog and an example of FFT smoothing.

FFT Filter **FFT Filter→Low Pass** This command allows to filter the high frequencies of a signal. See the filtering section for more details. A dialog box will be opened in which you can select the curve to filter and the cut-off frequency of the filter.

This command creates a new table with the filtered data, and a new curve will be added on the current plot. See Section 3.2 of the Chapter 3 for details.

FFT Filter→High Pass This command allows to filter the low frequencies of a signal. See the filtering section for more details. A dialog box will be opened in which you can select the curve to filter and the cut-off frequency of the filter.

This command creates a new table with the filtered data, and a new curve will be added on the current plot. See Section 3.2 of the Chapter 3 for details.

FFT Filter→Band Pass This command allows to filter the low and high frequencies of a signal. See the filtering section for more details. A dialog

box will be opened in which you can select the curve to filter and the cut-off frequency of the filter.

This command creates a new table with the filtered data, and a new curve will be added on the current plot. See Section 3.2 of the Chapter 3 for details.

FFT Filter→Block Band This command allows to keep the low and high frequencies of a signal. See the filtering section for more details. A dialog box will be opened in which you can select the curve to filter and the cut-off frequency of the filter.

This command creates a new table with the filtered data, and a new curve will be added on the current plot. See Section 3.2 of the Chapter 3 for details.

Interpolate Performs an interpolation. The curve must have enough data points to compute the interpolated points, if not a warning message will be prompted out.

The methods available to perform the interpolation are *Linear* (the curve must contain at least 3 points), *Cubic Spline* (the curve you analyse must contain at least 4 points, if not a warning message will be prompted out, *Non-rounded Akima spline* (the curve you analyse must contain at least 5 points). See the Section 3.7 of the Chapter 3 for a comparison of the differents methods.

This command creates a new curve on the current plot, and a new table.

FFT Performs a forward or inverse FFT transform of the selected curve. The inverse FFT transform of a forward transform will result in a data set identical to that used for the forward transform.

Quick Fit

Quick Fit→Fit Linear Performs a linear fit of the selected curve. The results will be given in the Log panel

Quick Fit→Fit Polynomial Opens the Polynomial Fit dialog, allowing you to choose the curve to fit, the order of the polynomial function to use, the number of points of the resulting curve and the abscissae limits for the fit.

Quick Fit→Fit Exponential Decay

Quick Fit→Fit Exponential Decay→First Order Opens the Exponential Fit dialog, allowing you to choose the curve to fit and the initial guesses for the fit parameters.

Quick Fit→Fit Exponential Decay→Second Order Opens a dialog, allowing you to choose the curve to fit and the initial guesses for the fit parameters.

Quick Fit→Fit Exponential Decay→Third Order Opens a dialog, allowing you to choose the curve to fit and the initial guesses for the fit parameters.

Quick Fit→Fit Exponential Growth Performs an exponential growth fit of the selected curve.

Quick Fit→Fit Lorentzian Performs a lorentzian fit of the selected curve. It can be used to obtain a correlation equation of a bell shaped data set (see Section 3.6.2.5 for details).

Quick Fit→Fit Gaussian Performs a gaussian fit of the selected curve.It can be used to obtain a correlation equation of a bell shaped data set (see Section 3.6.2.4 for details).

Quick Fit→Fit Bolzmann (Sigmoïdal) Performs a fit to a bolzmann function of the selected curve. It can be used to obtain a correlation equation of a S shaped data set. (see Section 3.6.2.3 for details).

Quick Fit→Fit Multipeak

Quick Fit→Fit Multipeak→Gaussian Performs a fit to a sum of N gaussian functions of the selected curve. (see Section 3.6.3 for details).

Quick Fit→Fit Multipeak→Lorentzian Performs a fit to a sum of N lorentz functions of the selected curve. (see Section 3.6.3 for details).

Fit Wizard (CTRL+Y) Performs a fit of the selected curve. This opens the general dialog for the fitting of curves. See the Section 3.6.1 for a tutorial on this command. Some default parameters can be modified with the Preferences command, see the Section 3.6.4 for details

5.9 The Table Menu

This menu is only active when a table is selected. For a general presentation of the tables, refer to the Section 1.3.1.

Set Column as These commands are used to define the kind of data which is stored in the different columns of a table. They can also be accessed with the right mouse button when a column is selected in a table.

Set Column as→X Define the selected column as abscissae for the plots. You can define more than one column as X-values in a tables, they will be referenced as X1, X2, etc.

Set Column as→Y In the case of 2D plots, this command defines the selected column as Y-values for the plots. In the case of 3D plots, Y columns can be used as the second abscissae.

Set Column as→Z In the case of 3D plots, Z columns will be used as plotted values.

Set Column as→X Error Define the selected column for use as error bars width for abscissae. Note that the column is not related to a specific X column, you will have to specify the link to specific X values when the plot will be built.

Set Column as→Y Error Define the selected column for use as error bars for Y-values. Note that the column is not related to a specific Y column, you will have to specify the link to specific Y values when the plot will be built..

Set Column as→None The selected column can be used in different ways in several plots (as X values, Y values, etc).

Fill Selection With This command is used to fill the selected column with special values. It can be applied to a limited selection of cells. These commands does not assign formulas to cells, they just fill in the cells with values.

Fill Selection With→Row Numbers The filling is done with the number of the corresponding rows.

Fill Selection With→Random Values The filling is done with random values between 0 and 1.

Show Comments If you select this command, the *Comment* field of the columns will be shown under the names of the columns. The name of the command will then change to *hide comments*. This command applies only to the selected table. See the Section 1.3.1 for more details.

Hide Controls If you select this command, the *Parameters* part of the table will be shown. The name of the command will then change to *hide controls*. This command applies only to the selected table. See the Section 1.3.1 for more details.

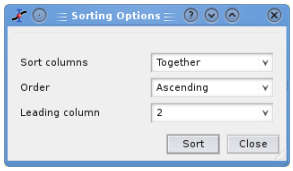
Formula Edit Mode If you select this command, the formula used in the different columns of the table will be shown. This command applies only to the selected table. In this mode, the formula assigned to each cell can be viewed and edited. This allows to use different formulas on each row of a column. Then, you can switch back to the normal mode and used the Recalculate command to view the numbers resulting from these formulas.

Edit Column Description This command is just a shortcut to the *Description* tab of the table. See the Section 1.3.1 for details.

Change Type & Format This command is just a shortcut to the *Type* tab of the table. See the Section 1.3.1 for details.

Clear Table Removes all the values of the selected table. There is no confirmation window for this command, but you can use the Undo command to cancel.

Sort Table This command is used to sort the table. If you choose the option separately, only the selected column is sorted. If you choose together, all the columns are sorted based on the specified leading column.



This command is used to fill the selected column with the values resulting from a mathematical formula. This command will open the Formula tag in the properties dialog of the selected table.

The available mathematical functions (assuming you are using the default scripting language, muParser) are listed in the Section 4.1. The special function *col(x)* can be used to access to the values of the column x, where x can be the column's number (as in *col(2)*) or its name in doublequotes (as in *col("time")*). You can also get values from other tables using the function *tablecol(t,c)*, where t is the table's name in doublequotes and c is the column's number or name in doublequotes (example: *tablecol("Table1","time")*).

The variables `i` and `j` can be used to access the current row and column numbers. Similarly, `sr` and `er` represent the selected start and end row, respectively.

Using Python as scripting language gives you even more possibilities, since you can not only use arbitrary Python code in the function body, but also access other objects within your project. For details, see Section 4.2.

If you make some changes in the table, the values are not computed again. You have to explicitly tell SciDAVis to recalculate individual cells or whole columns or rows by selecting Recalculate command from their context menu or pressing CTRL+Return.

Assign Formula **Recalculate** When you fill a column (named for example 'C1') with the results of a formula (by using the Assign Formula command), the values of the column are calculated only once when you define the formula. If your formula depends on values of another column (name for example 'C2'), the values of 'C1' are not updated if you modify the values in 'C2'. This command is used to recalculate the values of the selected column.

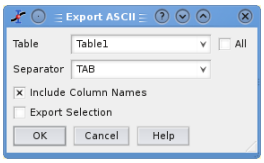
Add Column Adds a new column in the table. Whatever the selected column, the new one will be inserted at the right of the table after the last column. If you want to insert a column between two existing ones, select the column and use *Insert Empty Columns* from the context menu. A new column will be created on the left of the selected column.

Dimensions Allows to define the number of columns in the table. Be carefull if you decrease the number of columns in a table, a number of columns will be removed and the data will be lost.

Allows to define the number of rows in the table. Be carefull if you decrease the number of rows in a table, a number of rows will be removed and the data will be lost.

Go To Cell Defines the active line in the selected table.

Export Ascii This command can be used to export the selected table to an ascii text file. If you check the option *All*, you will have to choose a directory in which one text file will be created for each table, the name of the files being the one of the tables.



Convert to Matrix This command is used to convert a table into a matrix. It is mainly used to import data from files: the first step import data in a table, and the second one is the conversion of the table in a matrix.

5.10 The Matrix Menu

This menu is only active when a matrix is selected. See Section 1.3.2 for details on matrices.

Hide Controls This command opens a dialog window which is used to specify the size of a matrix. It can also be used to specify the X and Y ranges which will be used as axis ranges for a 3D-plot of the matrix data. See Section 1.3.2 for details.

Set Coordinates This command is just a shortcut to the *Coordinates* tab of the properties dialog of the selected matrix. See Section 1.3.2 for details.

Dimensions This command opens a dialog window which is used to specify the size of a matrix.

Set Display Format This command is just a shortcut to the *Format* tab of the properties dialog of the selected matrix. See Section 1.3.2 for details.

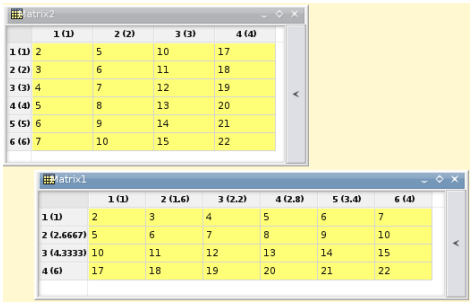
Assign Formula This command is just a shortcut to the *Formula* tab of the properties dialog of the selected matrix. See Section 1.3.2 for details.

You can fill in a matrix with the results of a function z=f(i,j) in which i and j are the row and column numbers. If you have defined X-values and Y-values with the Set Coordinates command You can use x and y as parameters for the function. The functions can be written on several lines, and the intrinsic functions which are available are listed in the Section 4.1.

Recalculate This command apply the formula assigned to the matrix (with the Assign Formula command) to all the cells of the selected matrix. The values which may have been entered in some cells will be overwritten.

Clear Matrix Set all the values of the matrix to 0. There is no confirmation window but you can use the Undo command to cancel this command. The formula and the coordinates which may have been entered are not detroyed by this command.

Transpose Replace the selected matrix with the transposed one. If you want to keep a copy of the pristine matrix, use the Duplicate command before transposing. The matrix doesn't need to be square. Beware that the coordinate are not transposed.



Mirror Horizontally Mirror the values of the selected matrix horizontally. If you want to keep a copy of the pristine matrix, use the Duplicate command before mirroring. The coordinate are not mirrored.

Mirror Vertically Mirror the values of the selected matrix vertically. If you want to keep a copy of the pristine matrix, use the Duplicate command before mirroring. The coordinate are not mirrored.

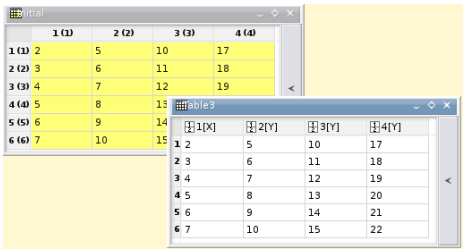
Import Image This command is similar to Import Image command except the fact that it replace the selected matrix with the image matrix instead of creating a new matrix.

Go To Cell You can specify the line and column number of a cell.

Invert Inverse the selected matrix. If you want to keep a copy of the pristine matrix, use the Duplicate command before mirroring.

Determinant Compute the determinant of the selected matrix. The result is given in the Results Log command

Convert to Table Convert the selected matrix in a table. The pristine matrix is kept and a new table is created. The coordinates are lost.



5.11 The Format Menu

This menu is only active when a plot is selected. Refer to the Chapter 2 for a tutorial on the formatting of 2D or 3D plots.

Plot *2D plot*

This command is used to set some general graphic parameters of the different layers and of the curves. Refer to the Section 2.1.4.1.

In addition, you can specify some global parameters of the plot with the format dialog with the *General* tab selected. The canvas is the area defined by the axis, you can draw a box around this canvas and define a background color for this canvas. The background area is the global drawing area, you can also define a color border and a background color for this area. The margin parameter controls the distance between the drawing area limit and the canvas. If you want to modify the margin between the window limits and the drawing area, you must modify the layer parameters (manualy with the mouse or with the Arrange Layers command.

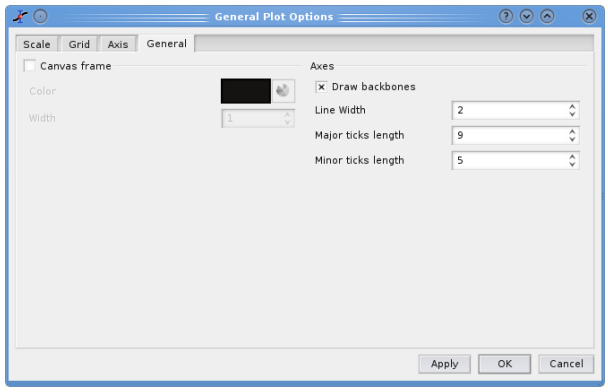


Figure 5.31: 2D plot options dialog: General settings.

The parameters in the *Axis* group allow to modify the linestyle of the axes and of the ticks.

3D plot

In the case of a surface plot, this command opens the surface plot options with the *General* plot options tab selected. In this case the aspect ratio of the plot can also be modified. The default behaviour is to use the perspective to compute the 3D plot. If you choose to check the *Orthogonal* check box, the plot will use vertical Z axis whatever the view angle of the plot.

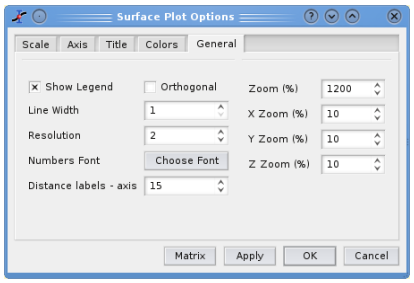


Figure 5.32: Surface plot options: general settings.

Scales *2D plot*

Opens the format plot dialog with the scales tab selected. It allows to customize the ranges of the differents axes. It must be reminded that any modification in the table or in the plotted curves will result in a reset of these scales to the default values.

In the case of a surface plot, this command opens the surface plot options with the scales options tab selected.

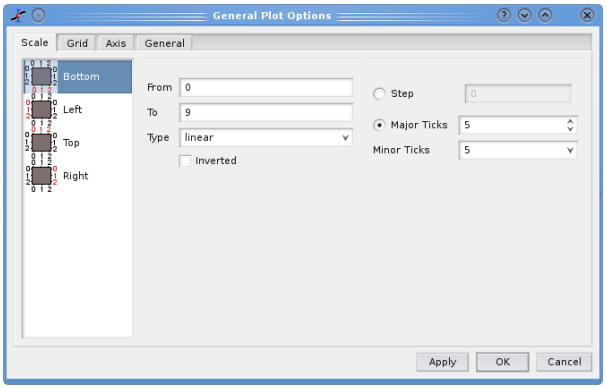


Figure 5.33: Plot options dialog: scales settings.

In this tab, you can also set the number of ticks used for each axis. This can be done in two ways: you can set the number of labels which are used for the whole scale. Whatever the number you enter, SciDAVis will use a value which leads to a pretty plot: for example, if you enter 7 ticks for a 0..100 scale, SciDAVis will use 10 major ticks from 10 to 10. If you want to fix non classical values, you can select the step method.

3D plot

The first tab is used to modify the X, Y and Z ranges. It allows also to specify the number of labels on the axis and the number of secondary ticks.

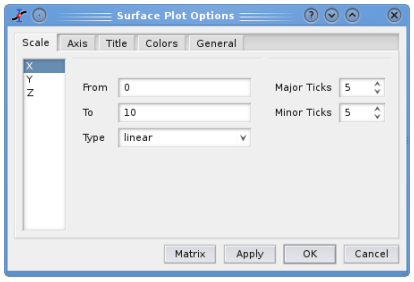


Figure 5.34: Surface plot options: scales settings.

Axes *2D plot*

Opens the format plot dialog with the axes tab selected. It allows to customize the settings for the different axes such as the size and color of axes and ticks, the label of the axes, etc. The third tab is used to modify the setting of the different axis. You must select the axis that must be customized in the right window. The label of the axis can be modified in the title window, see the text dialog section for more details.

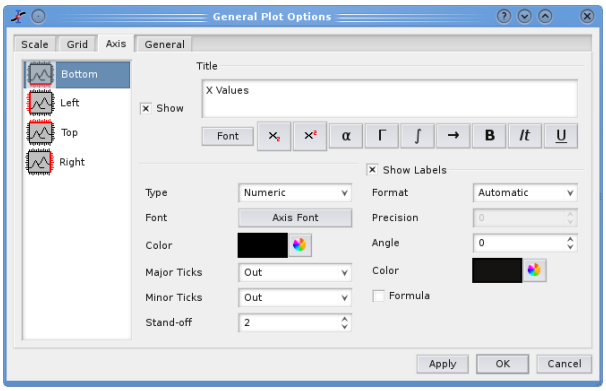


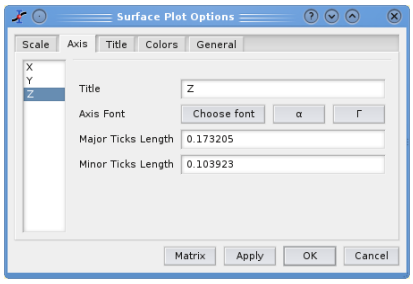
Figure 5.35: General plot options dialog: the axis tab.

3D plot

In the case of a surface plot, this command opens the surface plot options with the axis options tab selected.

The second tab defines the main parameters of the three axis: the axis label and its font, and the length of the ticks. This length is defined in the same units as the

range of the axis. If something is changed in the scales of the graph, the length of the ticks is re-calculated by SciDAVis. The font button allows to modify only the font used for the label, if you want to customize the font of the numbers used for the axis, you must used the fifth tab.



Grid *2D plot*

Opens the format plot dialog with the grid tab selected. It allows to add and customize grid lines on the different axes. The grid tab is used to draw grid lines on the plot. The frequency of the lines are related to the number of label and major ticks set with the *Scale* tab.

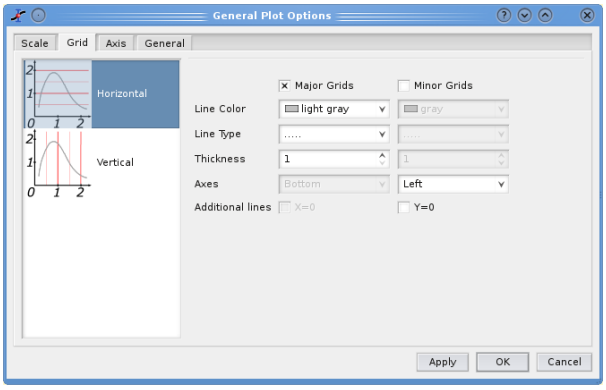


Figure 5.36: General plot options dialog: the grid tab.

If the selected plot is a surface plot, this menu item is not showed.

Title *2D plot*

Opens a *text dialog*, allowing you to modify the title of the plot and its properties (color, font, alignement). See the Section 2.6.1.

3D plot

In the case of a surface plot, this command opens the surface plot options with the title options tab selected. You can not add subscripts, superscripts, bold characters, etc in your title as you can do it for 2D plots.

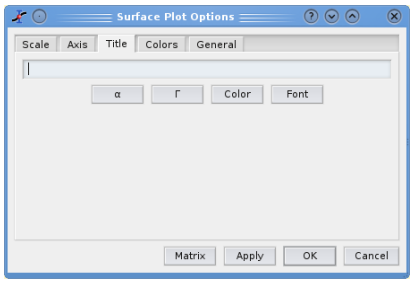


Figure 5.37: Surface plot options dialog: the title tab.

5.12 The Window Menu

Additionaly to the items listed bellow, this menu will also display a list with the first ten windows created in the workspace. These windows can be made active or can be shown if they are hidden, by selecting their name from the list. If your project contains more then ten windows, you must use the Project explorer in order to perform these operations.

Cascade Arranges the visible windows in the project in a cascading style.

Tile Tiles the visible windows in the project.

Next (F5) Makes the next visible window in the workspace stack the active window.

Previous (F6) Makes the previous visible window in the workspace stack the active window.

Rename Window Opens a dialog allowing to change the title of the active window.

Duplicate Clonates the active window.

Window Geometry... Opens a dialog allowing to change the size and the position of the active window. The size of the plot will be adapted to the new window size.

Hide Window Hides the active window. A hidden window can be made visible again via the Project explorer.

Close Window (Ctrl-W) Closes the active window. You will be prompted out a question dialog asking you to confirm the operation, if you checked this option in the Preferences dialog ("Confirmations" tab).

5.13 The Help Menu

Help If you have configured the help folder with the **Choose Help Folder**, this command will launch the *qt-assistant* help browser. The last version of scidavis manual can be obtained from https://sourceforge.net/projects/scidavis/.

Choose Help Folder Let you define the folder which contain the SciDAVis manual. The manual should be in html version.

Scidavis Homepage This command launch the default browser of your system with the home page of the SciDAVis project opened.

Search for Updates Let SciDAVis look for updates. You should have an effective internet connection to use this command.

Download Manual Download the last version of the SciDAVis manual from the https://sourceforge.net/projects/scidavis/ site.

Translations Look for available translation files on the https://sourceforge.net/projects/scidavis/ site.

Visit Scidavis Forum This command launch the default browser of your system with the forum page of the SciDAVis project opened.

Report a Bug This command launch the default browser of your system with the bug page of the SciDAVis project opened.

About Scidavis Open the window which shows the version and the credits of SciDAVis.

Chapter 6

The Toolbars

All toolbars can be moved and docked to a more convenient location (left, right or bottom sides of the application window) or on the desktop (outside the main window) by drag-and-drop, using their left side handle. The toolbars are automatically enabled/disabled depending on the currently active window: for example if the current window is a table, the Table toolbar will be enabled and all the other toolbars will be automatically disabled.

The same approach is used for showing/hiding the toolbars: if there are no more visible tables in the workspace, the Table toolbar will be automatically hidden and will be shown again when the users adds a new table into the project. A toolbar can be manually shown/hidden by the user, at any time, by right-clicking on the main window menu area and checking/unchecking the corresponding box in the pop-up menu.

6.1 The File Toolbar

The *File Toolbar* allows to access commands mainly from the File menu. Refer to this section for a more complete description of these commands.

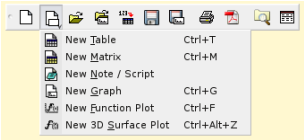


Figure 6.1: The SciDAVis File Toolbar

Icon	Command	Key	Description
	New→New Project command	CTRL+N	Create a new project.
	New command		Access to the *New* sub-menu.
	New→New Table command	CTRL+T	Create a new table.
	New→New Matrix command	CTRL+M	Create a new matrix.
	New→New Note/Script command	CTRL+ALT+N	Create a new note window, this object can be used as a calculator or to use scripts.
	New→New Graph command	CTRL+G	Create a new empty 2D plot.
	New→New Function Plot command	CTRL+F	Creates a new plot based on a function Y=f(X).
	New→New 3D Surface Plot command	CTRL+ALT+Z	Creates a new 3D plot based on a function Z=f(X,Y).
	Open command	CTRL+O	Opens an existing SciDAVis project file.
	Open Template command		Opens an existing template SciDAVis project file.
	Save Project command	CTRL+S	Saves the current project.
	Save as Template command		Saves the current project as a template.
	Import Ascii command		Imports an ASCII file into one or multiple tables.
	Print command	CTRL+P	Print the active window.
	Export to PDF command		Export to PDF.
	Project Explorer command	CTRL+E	Show or hide the project explorer.
	Results Log command		Show or hide the results window.

Table 6.1: File toolbar commands.

6.2 The Edit Toolbar

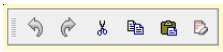


Figure 6.2: The SciDAVis Edit Toolbar

Icon	Command	Key	Description
	Undo command	CTRL+Z	Undo the last command, this feature doesn't work for plot modifications.
	Redo command	CTRL+R	Redo the last command, this feature doesn't work for plot modifications.
	Cut Selection command	CTRL+X	Cut the current selection.
	Copy Selection command	CTRL+C	Copy the current selection.
	Paste Selection command	CTRL+V	Paste the current selection.
	Delete Selection command	del	Delete the current selection.

Table 6.2: Edit toolbar commands.

6.3 The Plot Toolbar.

This toolbar is only active when a table is selected. It allows the quick access to the commands of the Plot menu which are used for the creation of new plots.

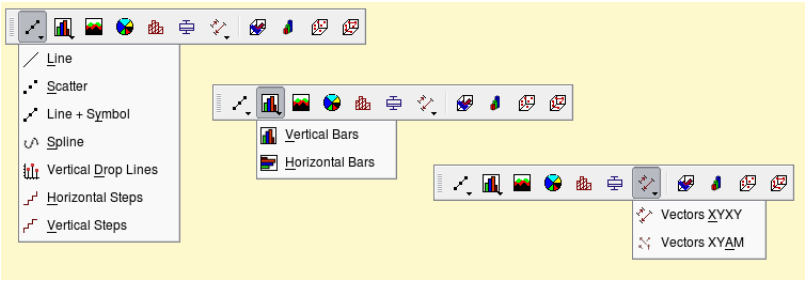


Figure 6.3: The SciDAVis Plot Toolbar with its different sub-menus

Icon	Command	Key	Description
	Access to the submenus for lines/Symbol plot types.		
	Line command		Build a graph with data plotted as lines
	Scatter command		Build a graph with data plotted as scatter of points
	Line+Symbol command		Build a graph with data plotted as lines with symbols
	Special Line/Symbol→Splines command		Build a graph with data plotted as smoothed lines
	Special Line/Symbol→Vertical Drop Lines command		Build a graph with data plotted as vertical drop lines
	Special Line/Symbol→Horizontal Steps command		Build a graph with data plotted as horizontal step lines
	Special Line/Symbol→Vertical Steps command		Build a graph with data plotted as vertical step lines
	Access to the sub-menu for columns and rows plots		
	Vertical Bars command		Build a graph with data plotted as columns
	Horizontal Bars command		Build a graph with data plotted as rows
	Area command		Build a graph with data plotted as lines with a filling of areas.
	Statistical graphs→Histogram command		Build a graph with data plotted as an histogram.
	Box Plot command		Build a graph with data plotted as an histogram.
	access to the sub-menu for vector plots.		
	Vectors XYXY command		Build a graph with data plotted as vectors defined by two points.
	Vectors XYAM command		Build a graph with data plotted as vectors defined by an origin and a direction.

Table 6.3: Plot toolbar commands

6.4 The Graph Toolbar.

This toolbar is only active when a plot window is selected. It allows the quick access to the commands of the Graph menu which are used for the modification of the plots and of the data points of the plots.

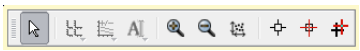


Figure 6.4: The SciDAVis Graph Toolbar with its different sub-menus

Icon	Command	Key	Description
	Disable Tools command		Comes back to the normal pointer mode, this is useful when you have select other modes of the plot window such as the data reader.
	Access to the layers commands.		
	Arrange Layers command	ALT+A	Arranges the different layers of the active plot window.
	Add Layer command	ALT+L	Adds a new layer to the active plot window, or remove a layer from the selected plot window.
			Access to the curves sub-menu.
	Add/Remove Curve command	ALT+C	Adds or removes curves to the active plot window.
	Access to the commands for addition of graphics objects to the current plot.		
	Add Text command	ALT+T	Add a new text element in the active plot.
	Zoom In command	CTRL++	Switches the active plot layer to the zoom mode.
	Zoom Out command	CTRL+-	Switches the active plot layer to the zoom mode.
	Rescale to Show All command	CTRL+SHIFT+R	Reset the zoom in order to show all the data.
	Screen Reader command		Switches the active plot layer to the **Screen Reader** mode.
	Data Reader command	CTRL+D	Switches the data display mode.
	Select Data Range command	ALT+S	Switches the active plot to the **Select Data Range** mode.

Table 6.4: Plot toolbar commands

6.5 The Table Toolbar.

This toolbar allows a quick access to the commands of the Table menu used to modify a table.

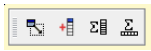


Figure 6.5: The SciDAVis Table Toolbar

Icon	Command	Key	Description
	Dimensions command		modify the number of rows and columns of the table
	Add Column command		add a new column to the table
	Statistics on Columns command		compute statistical parameters on selected columns
	Statistics on Rows command		compute statistical parameters on selected row

Table 6.5: Table toolbar commands.

6.6 The matrix plot Toolbar.

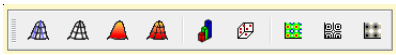


Figure 6.6: The SciDAVis matrix Plot Toolbar

6.7 The 3D Surfaces Toolbar.

Figure 6.7: The SciDAVis 3D Surfaces Toolbar

Icon	Command	Key	Description
	3D Wire Frame command		Draw a surface with the wireframe style.
	3D Hidden Lines command		Draw a surface with the mesh style (with hidden lines).
	3D Polygon command		Draw a surface with the polygons style.
	3D Wire Surface command		Draw a surface with the mesh+polygons style.
	Bar Style command		Changes the styles of the bars.
	3D Plots→Scatter command		Draw data points as a clouds of points in a 3D space.
	Contour+Color Fill command		Draw data points as a map with a color filling between isolines.
	Contour Lines command		Draw data points as a map with isolines.
	Gray Scale Map command		Draw data points as a map with a gray palette filling between isolines.

Table 6.6: 3D Plot toolbar commands.

Icon	Command	Key	Description
	Frame		Draw only the three axes.
	Box Plot		Draw the three axes and the 3D box around the plot.
	No Axes		Doesn't draw the axes nor the box.
	Front Grid		Draw a grid on the front panel. The position of this grid is the plan defined by $y=y_{min}$.
	Back Grid		Draw a grid on the back panel. The position of this grid is the plan defined by $y=y_{max}$.
	Left Grid		Draw a grid on the left panel. The position of this grid is the plan defined by $x=x_{min}$.
	Right Grid		Draw a grid on the right panel. The position of this grid is the plan defined by $x=x_{max}$.
	Top Grid		Draw a grid on the top panel. The position of this grid is the plan defined by $z=z_{max}$.
	plot -> Floor Grid		Draw a grid on the bottom panel.
	Floor Grid		Draw a grid on the bottom panel. The position of this grid is the plan defined by $z=z_{min}$.
	Perspective		Enables/Disables the 3D perspective mode.
	Reset Rotation		Resets the rotation of the 3D plot to the default values.
	Autoscale		Finds the best layout of the 3D plot fitting the window size. It readjusts the length of the axis ticks to a default value.
	Bar Style		If the active 3D plot is a 3D histogram, this command is used to modify the style of the bars.
	Dots		If the active 3D plot is a 3D scatter, this command is used to modify the style of the data points to dots.
	Cones		If the active 3D plot is a 3D scatter, this command is used to modify the style of the data points to cones. It is then possible to modify the drawing parameters of the cones by double clicking on the plotting area.
	Cross Hairs		If the active 3D plot is a 3D scatter, this command is used to modify the style of the data points to cross-hairs. It it then possible to modify the drawing parameters of the crosses by double clicking on the plotting area.
	3D Wire Frame		If the active 3D plot is a 3D surface, this command is used to modify the style of the surface to a simple wireframe.
	3D Hidden Lines		If the active 3D plot is a 3D surface, this command is used to modify the style of the surface to a wireframe. A computa-

Appendix A

Appendix

A.1 Credits and License

SciDAVis

Program copyright:
2004-2007 Ion Vasilief ion_vasilief@yahoo.fr
2006-2009 Tilman Hoener zu Siederdissen thzs@gmx.net
2006-2009 Knut Franke Knut.Franke@gmx.de
2012-2018 Russell Standish hpcoder@hpcoders.com.au
2013-2015 Dmitriy Pozitron dpozitron@users.sf.net
2016 Arun Narayanankutty narunlifescienc@users.sf.net
2017-2018 Miquel Garriga gbmiquel@users.sf.net
2017 Alexander Ploumistos alxpl@users.sf.net
Documentation copyright:
2004-2007 Ion Vasilief ion_vasilief@yahoo.fr
2006-2009 Roger Gadiou Roger.Gadiou@orange.fr
2006-2009 Knut Franke Knut.Franke@gmx.de
2017-2018 Fellype do Nascimento fellypao@yahoo.com.br
2017 Miquel Garriga gbmiquel@users.sf.net

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts.

A.1.1 GNU Free Documentation License

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

A.1.1.1 Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.1.1.2 Applicability And Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed

to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A.1.1.3 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.1.1.4 Copying In Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of

added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.1.1.5 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five). State on the Title page the name of the publisher of the Modified Version, as the publisher. Preserve all the copyright notices of the Document.

Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

Include an unaltered copy of this License.

Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.1.1.6 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any

sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

A.1.1.7 Collections Of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.1.1.8 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

A.1.1.9 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

A.1.1.10 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this

License will not have their licenses terminated so long as such parties remain in full compliance.

A.1.1.11 Future Revisions Of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http:///www.gnu.org/copyleft/. Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by

A.2 How to obtain SciDAVis

The SciDAVis home page can be found at https://sourceforge.net/projects/scidavis/. Updates and news can be found there.

SciDAVis is distributed as a package with sources which have to be compiled. Compiled packages suitable for most Linux distributions and Windows can also be obtained. If you want to build SciDAVis from sources, don't forget first to download and install the Qt, QwtPlot3D, Qwt, liborigin, zlib, muParser, and GSL libraries/dependencies. And also Python, SIP and PyQt if you want to enable Python scripting in SciDAVis.

A.3 Requirements

In order to successfully use SciDAVis, you need the following libraries:

Qt >= 4.2 SciDAVis uses the Qt toolkit. Version 4.2 or above is needed. It provides the necessary dynamic libraries to run the SciDAVis binaries and important tools to compile it.

Qwt You need to install the Qwt library version 5.x. It is recommended to use the 5.2.3 version. Qwt must be compiled with Qt 4.x.

muParser You also need the muParser library 1.28 or later.

Qwtplot3d The 3D plots in SciDAVis make use of qwtplot3d library. Version 0.2.6 or 0.2.7 is required.

GSL Furthermore, the GNU Scientific Library (GSL) (1.8 or later) must be installed on your system.

Zlib Additionally, zlib >=1.2.3 is required. Since this is also a requirement of Qt, installing Qt should fullfill this requirement on most systems.

Python scripting If you want to use Python expressions and scripts, make sure you have the following additional dependencies installed: Python >=2.5, SIP >=4.5.2, PyQt >=4.2. Other versions as those indicated above may or may not work.

A.4 Installation from binary packages

SciDAVis distributes binaries (usually for Windows and MacOS) which can be downloaded from the SciDAVis page on Sourceforge. Many Linux distributions have SciDAVis binaries in their repositories. You can also find the latest SciDAVis version for some distributions at the openSUSE build service. SciDAVis can be built easily on Arch Linux and Slackware at AUR and SlackBuilds.org, respectively.

A.5 Compilation and Installation from sources

To compile SciDAVis from sources, download the latest source distribution from the SciDAVis page on Sourceforge. It comes as a tar.gz archive containing detailed build instructions. If you are interested in the current development, you can also obtain a snapshot from the SciDAVis GitHub repository.

Appendix B

Frequently asked questions

Q: *How can I visualize data from a text file?*
A: Go to the File menu and select the Import Ascii command command.If the file is not imported correctly, change the columns separator.The default columns separator is the TAB.

Q: *How can I plot data from a table (worksheet)?*
A: Click on the table header to choose the columns to plot and then right click. Chose the 'Plot' option from the pop-up menu and then the type of plot you want.You can also use the plot assistant: press 'CTRL+ALT+W' keys to show it, or go to 'View' menu -> 'Plot wizard'.

Q: *How can I export a plot to an image format?*
A: Right click in the plot window and chose the 'Export' option.

Q: *Can I export transparent images?*
A: Yes, ".png" images have transparent background. See the Export Graph→Current command command.

Q: *How can I export a text file?*
A: Go to the File menu and select the Export Ascii command command.

Q: *How can I choose a window using the project explorer?*
A: Double click on the window name will show the window maximized, even if it was hidden before.

Q: *How can I choose the data range from a plot curve, when doing a curve fit?*
A: Go to the Tools menu and use the Select Data Range command command. Click in the plot window and use the 'Up' and 'Down' arrows keys to select the curve to analyze. Keeping 'CTRL' button and 'Left' or 'Right' arrow keys simultaneously pressed permit to move the selected cursor and consequently to modify the data range.

Q: *Can I fit a plot curve using my own function?*
A: Go to the Analysis-plots menu and select the Fit Wizard command command. Define the function (myFunction=...), enter the initial guesses for the parameters, separated by comas, choose the fitting range and the number of iterations and click 'OK'

Q: *How can I visualize a pixel line profile from an image?*
A: Right click on the image you want to analyze and select the option 'View pixel line profile' from the pop-up menu. A dialog window opens and allows you to select

the number of pixels used for the analysis. Choose a value and click "OK". Then click on the image to select the start point and move your mouse to select an end point while keeping the left button pressed. When you release the left button a plot window appears, representing the pixel intensity versus pixel index.

Index

A
Analysis
    Results, 9
Arrows and Lines
    Add an arrow/line, 116

C
Calculator, *see* Note
Columns
    Assign formula, 17
Curve analysis
    Curve filtering, 44
        Band pass FFT, 46
        Block pass FFT, 47
        High pass FFT, 45
        Low pass FFT, 45
    Curve fitting
        Bolzmann function, 52
        Gaussian function, 53
        line, 51
        Lorentz function, 53
        Multi peak, 54
        Non linear function, 49
        Polynomial, 52
    FFT, 43
    Integration, 131
    interpolation, 56

D
Data
    Export to text file, 106

F
Filtering, *see* Curve analysis

H
Histograms, 30

L
Log Window, 9

M
Matrix, 4, 7
    Create a new matrix, 100
    Fill with a function, 138
Multilayers plot, 38
    Add a new layer, 118
    Organize the layers, 118

N
Note, 8

O
Options
    2D plot, 22
    Application, 108

P
Percentile, 30
Plot, 4, 8, 11
    Add a curve, 112
    Axis, 142
    Change default options, 23
    Create a new plot, 100
    Create from data, 11
    Create from function, 15
    Create with the assistant, 110
    Error bars, 113
    Grids, 143
    Layer, 8
    Options for pie-plots, 25
    Options for vector-plots, 27
    pie-plots, 25
    Plot a function, 114
    Remove a curve, 112
    Scales, 141

secondary axis, 14
Settings, 140
Title, 143
vector-plots, 26
Plot details
Layer options, 19
Options for lines and symbols, 20
Specification of X and Y series, 20
Project Explorer, 10

S
Scripting, 57
MuParser, 57
Python, 57
Mathematical functions, 64
Statistical Plot
Box plots, 28
Histograms, 30
Statistical plots, 28
Surface plot, 32
Axis, 142
Create a new surface plot, 101
Create from data, 34
Create from function, 33
Default options, 37
Options, 35
Scales, 141
Settings, 140
Title, 143

T
Table, 4, 5
Assign formula, 6
Columns
Fill with values, 137
Create a new table, 99
Labels, 6
Number format, 6
table
normalize columns, 6
sort columns, 6
Table analysis
Convolution, 49
Correlation function, 47
Deconvolution, 49
Text label
Add a text label, 115
Properties, 41

W
Whiskers, 29