The QtiPlot Handbook

Ion Vasilief, Roger Gadiou, and Knut Franke

August 25, 2010

Contents

List of Figures

List of Tables

Abstract

This document is a handbook for using QtiPlot, a program for two- and three-dimensional graphical presentation of data sets and for data analysis.

This manual is organized in several chapters:

-The first one describe the main concepts and terms which are used in QtiPlot.

- The second is a tutorial on how to obtain plots from different data sets. It is the one you need to read first to understand the basics of QtiPlot and to be able to draw plots.

- The three following chapters are descriptions of all the commands, buttons and dialogs used in QtiPlot. These chapters are the reference manual of QtiPlot.

- The two following chapters describe more deeply some specific possibilities of QtiPlot, that is the statistical and mathematical analysis of data, and the scripting.

Chapter 1

Introduction

1.1 What QtiPlot does

QtiPlot is a 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 or from analytical functions.

The projet has been created by Ion Vasilief in 2000, and he was the only programmer between 2000 and 2005. Since 2006, new contributors have joined Ion and the project is hosted by BerliOS Developper. The software aims to be a tool for analysis and graphical representation of data in the way of commercial software like Origin.

QtiPlot 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 ".qti". These project files may be opened using the command line, or using the File menu, or by using the *Open project* 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 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 Informations command.

When the application is launched, a new untitled project file is created consisting of a grey main window (the workspace) which may contains an empty window, depending on your preferences. The type of the initial window can be customized using the Preferences dialog. It may be a table, a matrix, a note or an empty 2D graph window. In order to be operational, the workspace must be populated with tables storing data sets, either by creating empty tables/matrices first (New -> New Table command) and

then filling them with data, or by importing ASCII files (Import -> Import ASCII... command), which automatically creates new tables.

The user can easily navigate through the objects of a project file using the project explorer 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 QtiPlot from the command prompt, you can supply the name of a project file:

```
qtiplot file_name.qti
```

Other file formats 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:

```
qtiplot 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 values of the Import -> Import ASCII... command dialog. These 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
- -c or --console: show standalone scripting window
- -d or --default-settings: start QtiPlot with the default settings
- -h or --help: show command line options

- -l=XX or --lang=XX: start QtiPlot in language XX ('en', 'fr', 'de', ...)
- -m or --manual: show QtiPlot manual in a standalone window
- -v or --version: print QtiPlot version and release date
- -x or --execute: execute the script file given as argument
- -X: execute the script file given as argument without displaying the user interface. Warning: 2D plots are not correctly handled in this functioning mode!

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 QtiPlot. 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.

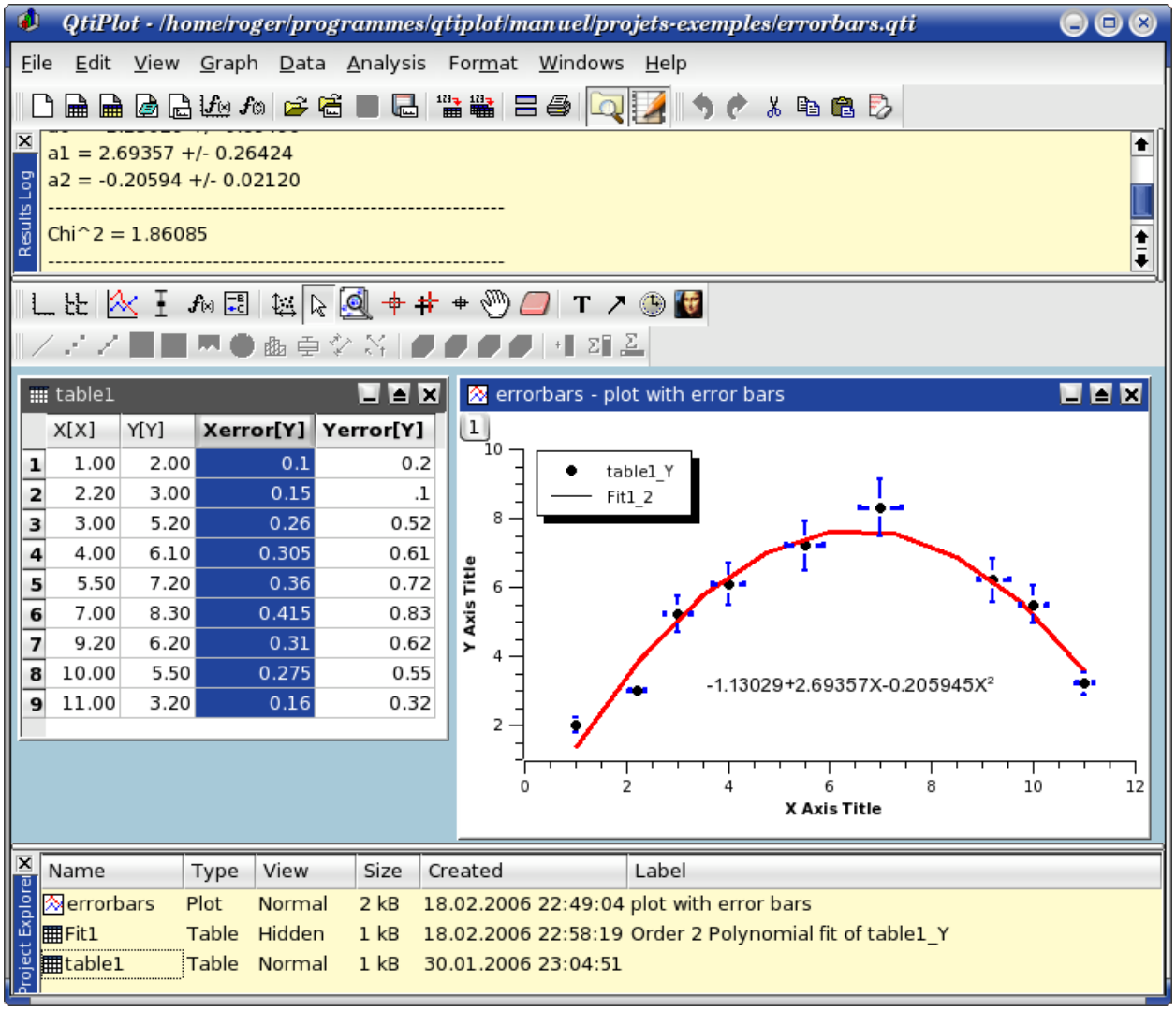


Figure 1.1: A typical QtiPlot session

There are numerous commands available in QtiPlot 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 containers which can be used are:

A Table 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.

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 -> Import ASCII... command. You can also enter each value from the keyboard. The last way to enter your data is to fill the table with the results of a mathematical function (Set Column Values... command from the Table menu)

A 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 Dimensions... 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 -> 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=(i,j) in which i and j are row and column numbers (Set Values... 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. You can also 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, but is really far more powerfull than that. It can be used as a calculator, for executing single commands and for writing scripts.

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 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.

Since the version 0.8.5, 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 QtiPlot 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 View menu. You can resize a table in terms of rows and columns using the Table menu with Rows command or Columns command.

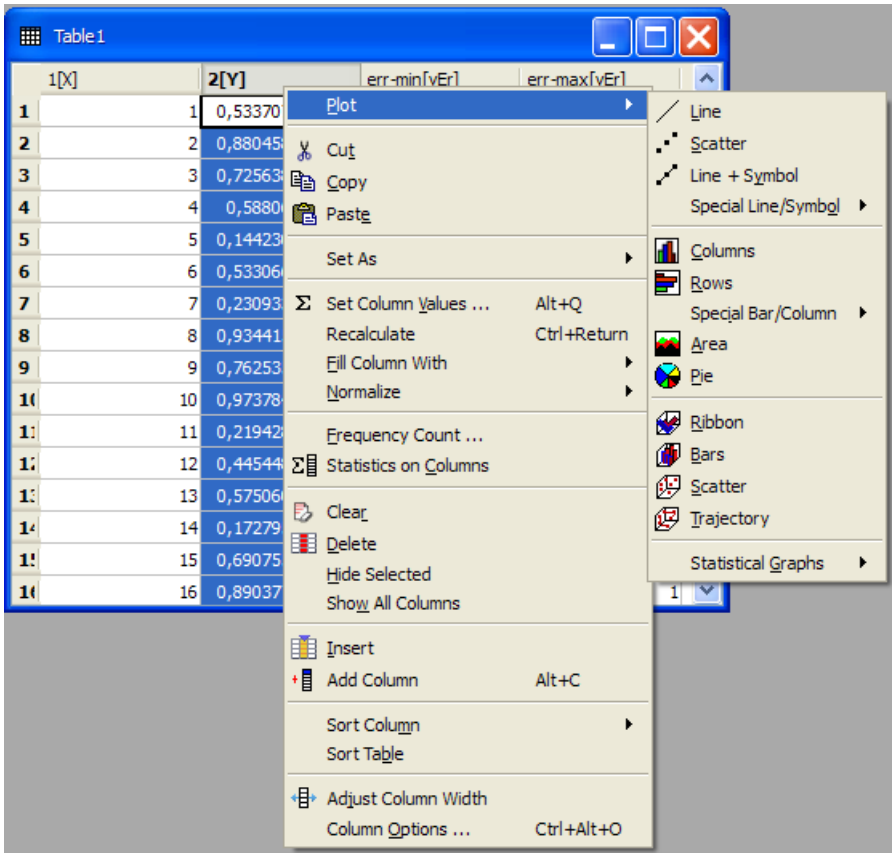


Figure 1.2: The QtiPlot table

Every column of the table has a label and can be assigned a format: numeric, text, date or time. 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 Column options dialog. To reach this dialog you simply have to double-click on the column label or to use the Column Options... command of the Table menu.

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, normalize, sort, view statistics and finally make plots out of your data. All these functions can be reached by right clicking on the column label or by using the Table menu.

Any other spreadsheet function: rename, duplicate, export, print, close can be reached via the context menu (right click anywhere in the table outside the column labels area).

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

You can import single or multiple ASCII files using the Import -> Import ASCII... command from the File menu. Of course you can 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 depending on two variables. This special table can be used to create 3D plots as well as 2D image/contour plots via the Plot 3D menu and the 3D plot toolbar. The difference between a table and a matrix is that matrices have a dual functioning mode: they can display data in a table form or they can display an image. Therefore matrices can be used as a basic image viewer and also as an image editor, since they implement some image manipulation funcions like: 90 degrees rotation, horizontal and vertical mirroring, etc...

In matrices there is no special column nor special row for X or Y labels or values. Nevertheless, you can specify an X-scale and an Y-scale with the Set Dimensions... command.

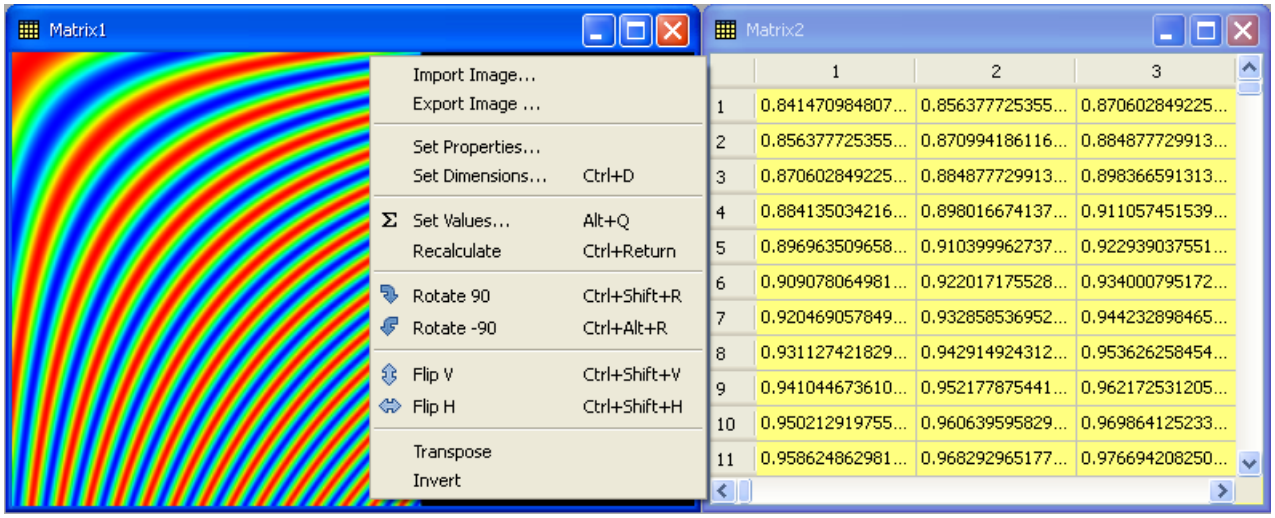


Figure 1.3: The QtiPlot matrix

The values which are stored in a matrix can be obtained from a function of the form z=f(i, j, x, y) with the Set Values... command, i and j beeing the column and row numbers and x and y the corresponding coordinates. They can also be read directly from an ASCII file with the Import -> Import ASCII... command or from an image file.

1.3.3 Plot Window

The plot window is the one in which the graphic is plotted. It contains at least one layer, which is the main container of the plot window. 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 example presented below shows a graph with two layers which have different geometries.

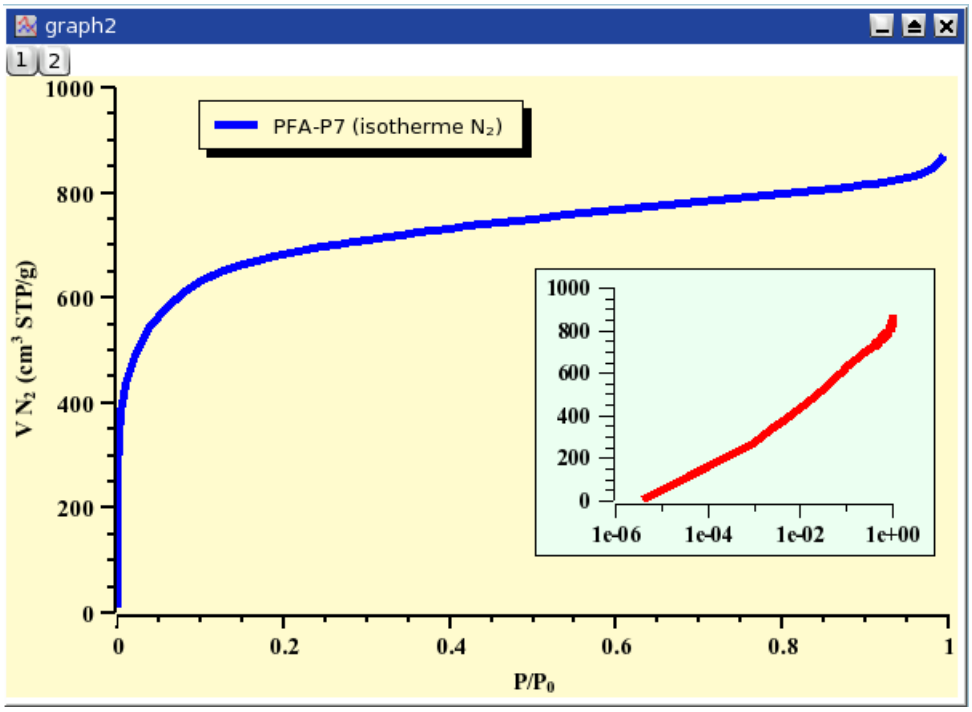


Figure 1.4: An example of QtiPlot 2D graph

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 Custom Curve Dialog. You can then add new curves to the plot, or 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.
- Text items, including the legend of the plot: this will open the Text Options Dialog which allows to customize the font of the label and the frame in which it is drawn.
- Arrow/Line items: this will open the Line Options Dialog.
- Image items: this will open a dialog allowing to customize the geometry and the position of the image.

A left click on a plot element selects it. You can deselect any element by presing the *Escape* key. A right click on a plot element pops-up a context menu allowing fast access to its properties dialog. Last but not least, you should know that QtiPlot provides

multiple selection for objects in a plot layer. In order to add an object to an existing selection keep the *Shift* key pressed and click on the element you want to add to the selection. Elements in a multiple selection can be moved and resized together with the mouse.

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. Evaluation of mathematical expressions and execution of code is done via a note's context menu, the Scripting menu or the convenient keyboard shortcuts. For information on expression syntax, supported mathematical functions and how to write scripts, see here.

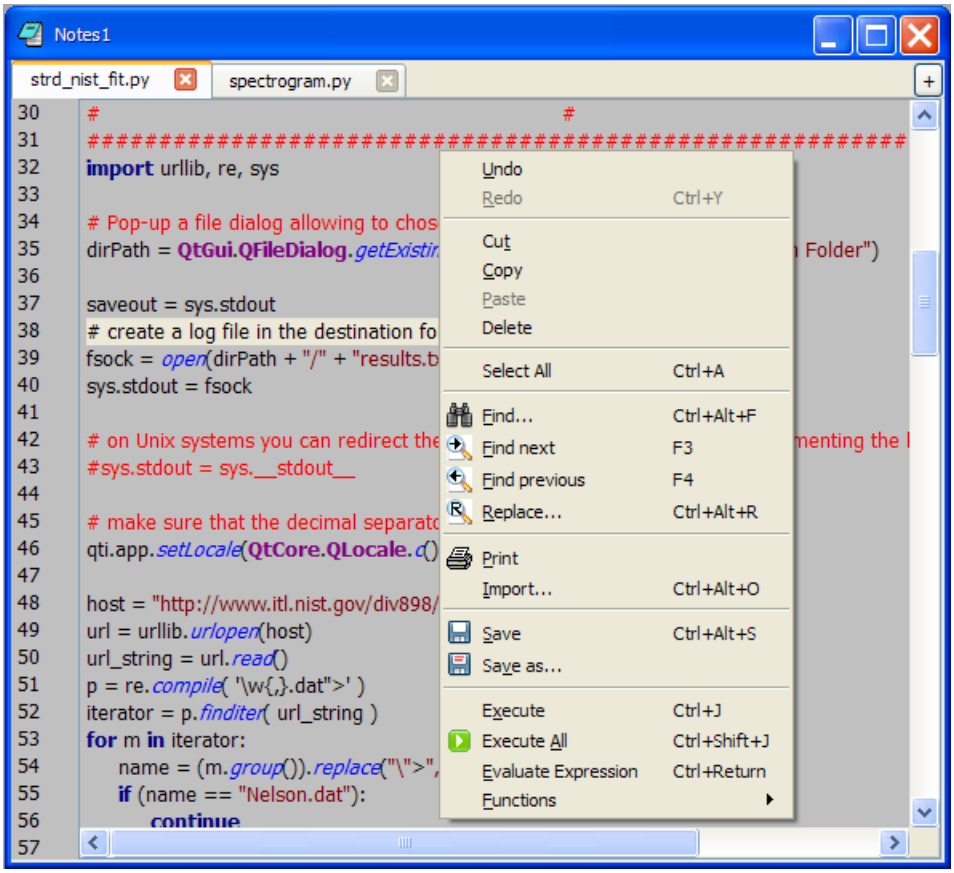


Figure 1.5: The QtiPlot Note Window

Note windows provide powerful text editor functionalities, particularly helpful when

writing scripts: customizable Python syntax highlighting, line numbers display, find and replace text, autocompletion suggestions for words that have more than two characters. You can trigger autocompletion using Ctrl+U. The colors used for syntax highlighting can be customized via the *Notes* tab in the Preferences dialog.

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.

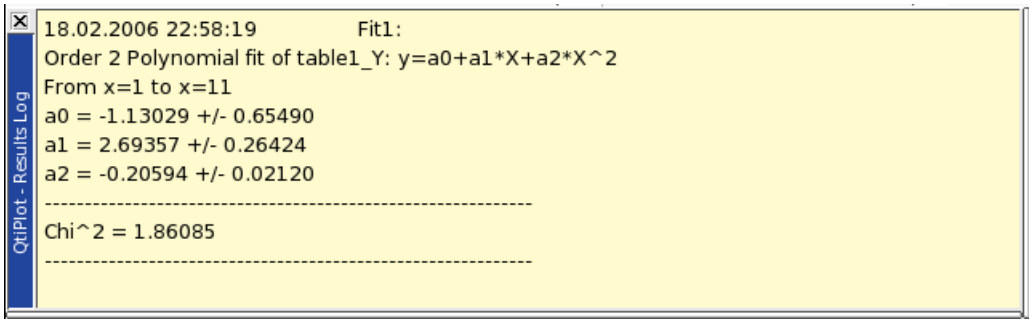


Figure 1.6: The QtiPlot Log window

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.

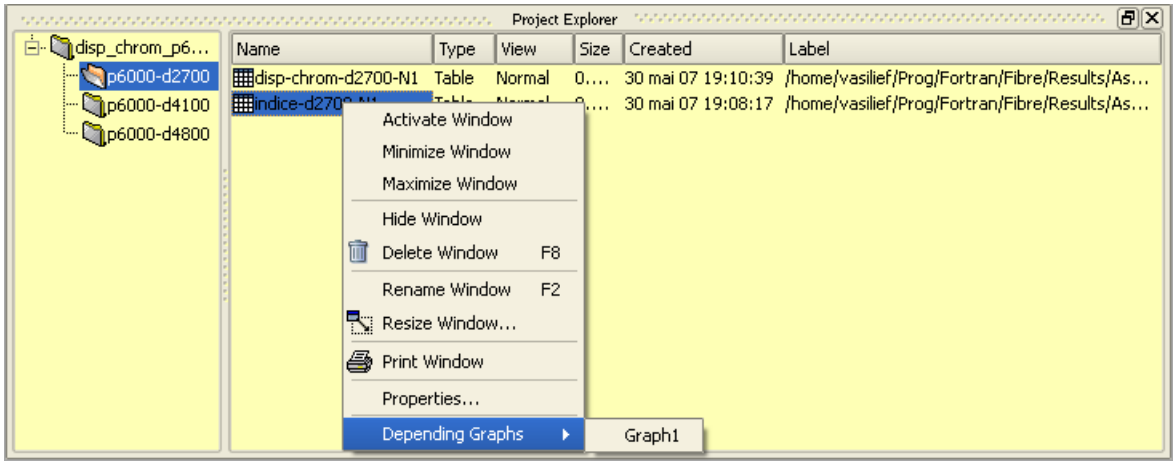


Figure 1.7: The QtiPlot Project Explorer

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 QtiPlot

2.1 2D plots

A 2D plot is based on curves which are defined by Y values as functions of X values. There are two 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 Column Options... 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.
- 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 Set Column Values... command. Then you can select the columns and use one command of the Plot menu to plot the data.

QtiPlot will create a new plot 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 -> 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 Rows command and Columns command from the Table menu. You enter your values, and obtain this table:

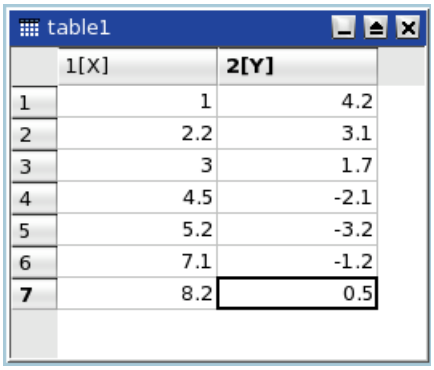


Figure 2.1: A simple 2D plot: the table.

The 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 preferences dialog. With the default options, you obtain the following plot:

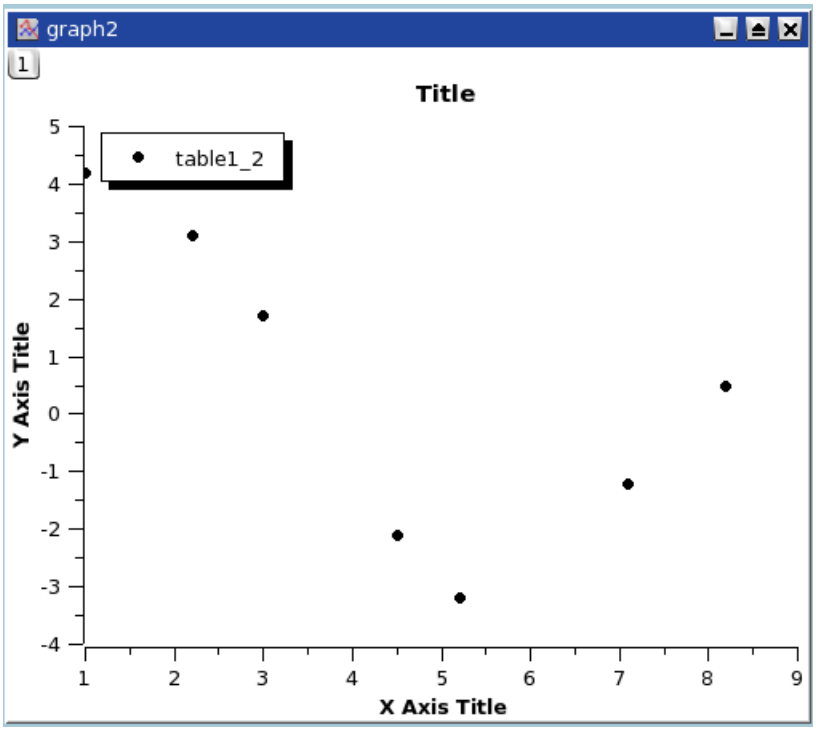


Figure 2.2: A simple 2D plot: the default plot.

You can then customize your plot. By double clicking on the points, you open the Custom curves dialog which is used to modify the symbols. Then a double-click on the axes opens the general plot options 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, 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. The final plot is:

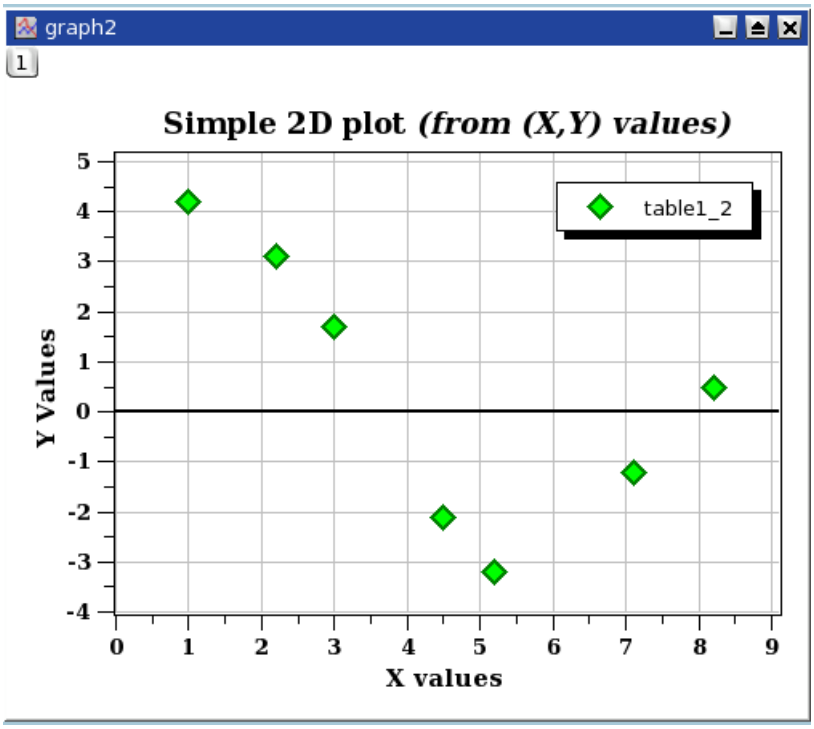


Figure 2.3: A simple 2D plot: the plot finished.

Finally, you have to save your project in a '.qti' 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

It is possible to use up to four axes for the data:

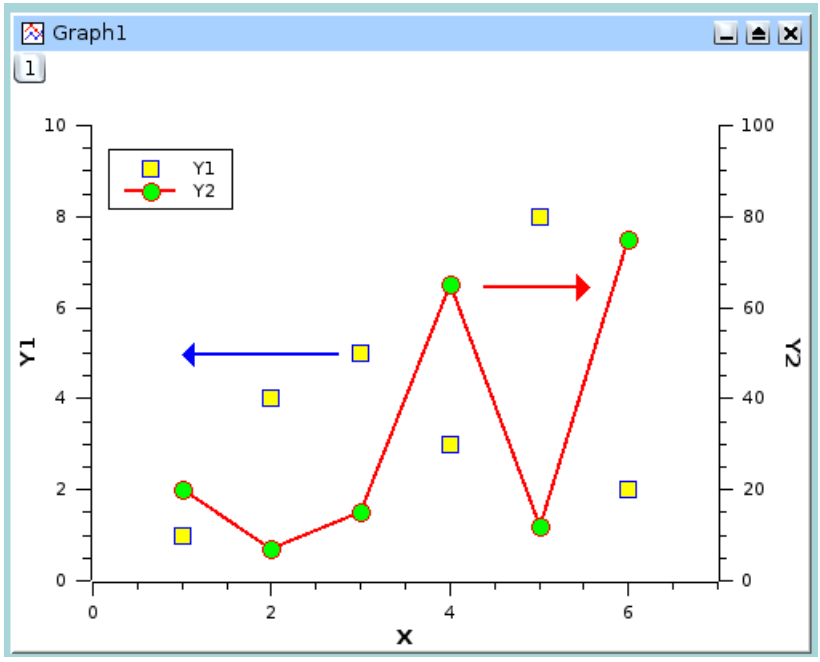
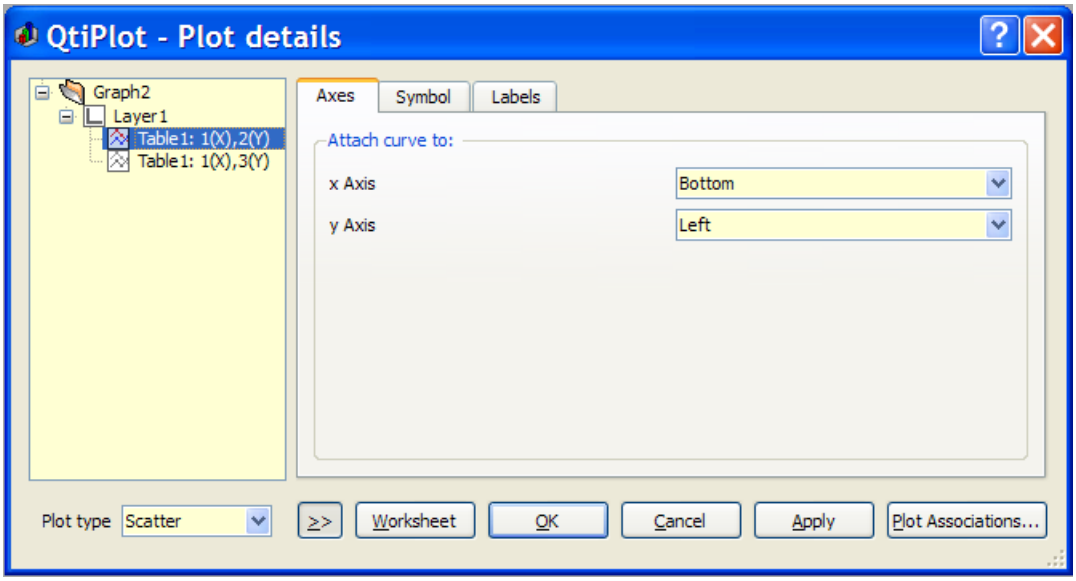


Figure 2.4: A 2D plot with two Y axis.

In addition to the customization which has been already described, the axes which are used for each curve were defined with the Custom Curves Dialog, and two 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.

This command will open the Add Function Curve dialog. 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. Beside classical Y=f(X) functions, parametric and polar functions can be defined.

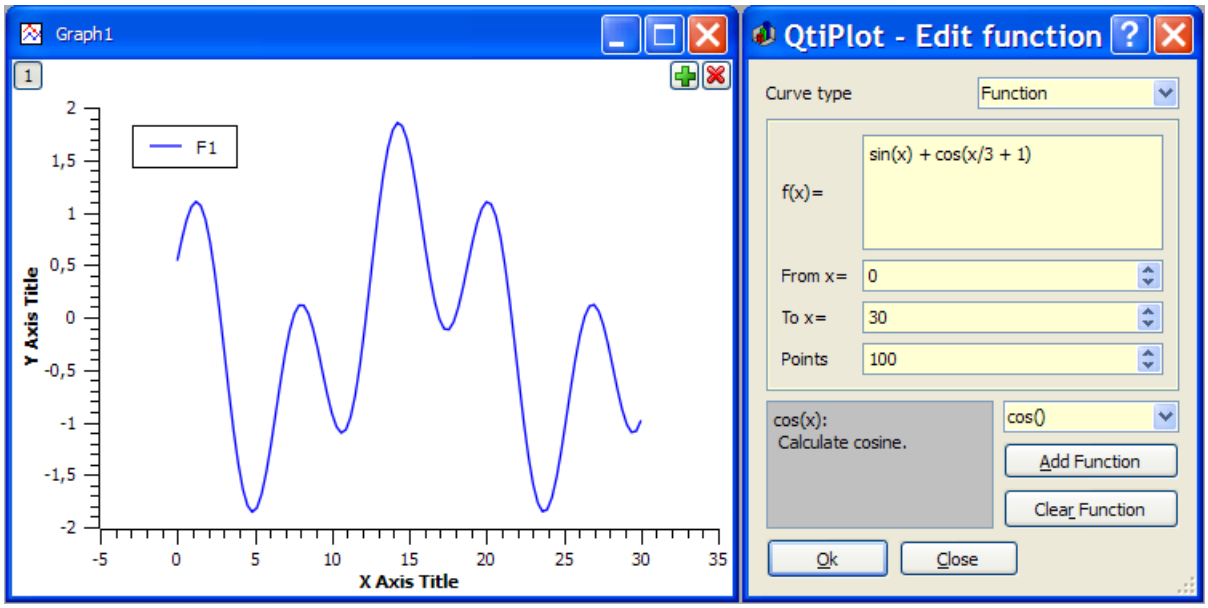


Figure 2.5: Direct plot of a function.

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 Set Column Values... command.

To obtain the same plot as in the previous example, you need to create a new table (key Ctrl-T), then select the first column and use the command Set Column Values... 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* and use 300 rows.

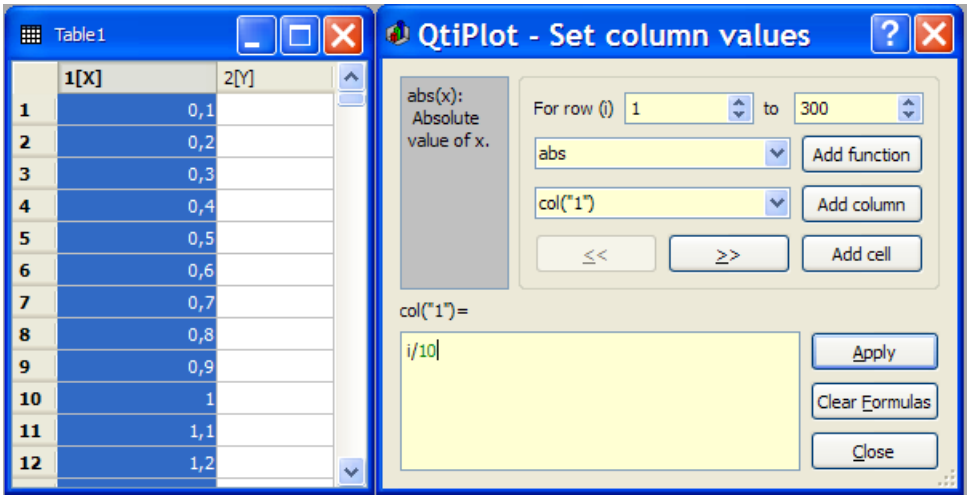


Figure 2.6: Function plot: filling of the X column.

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)*.

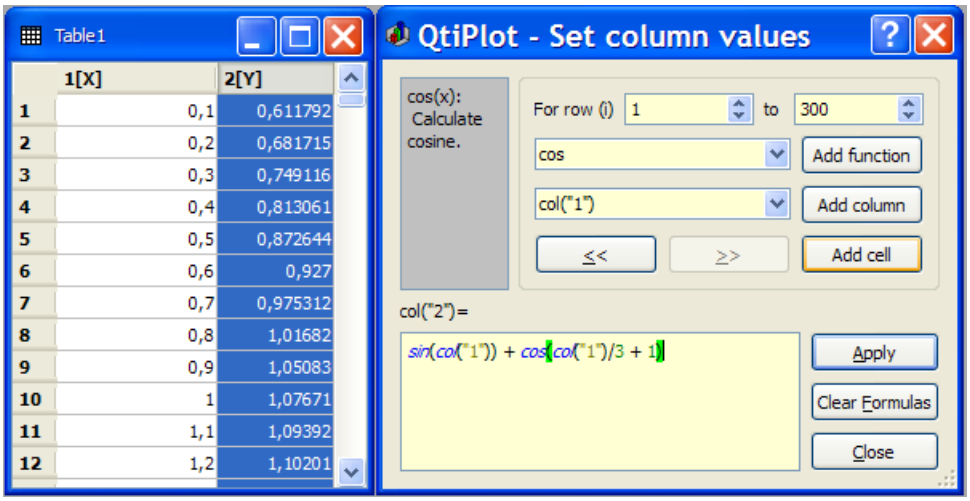


Figure 2.7: Function plot: filling of the Y column.

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

2.2 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. QtiPlot will consider that all the data present in the matrix are Z values, and the X and Y values can be defined as a 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 Surface 3D Plot command. This will open the corresponding dialog box and you will be able to define the mathematical expression of your function.

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

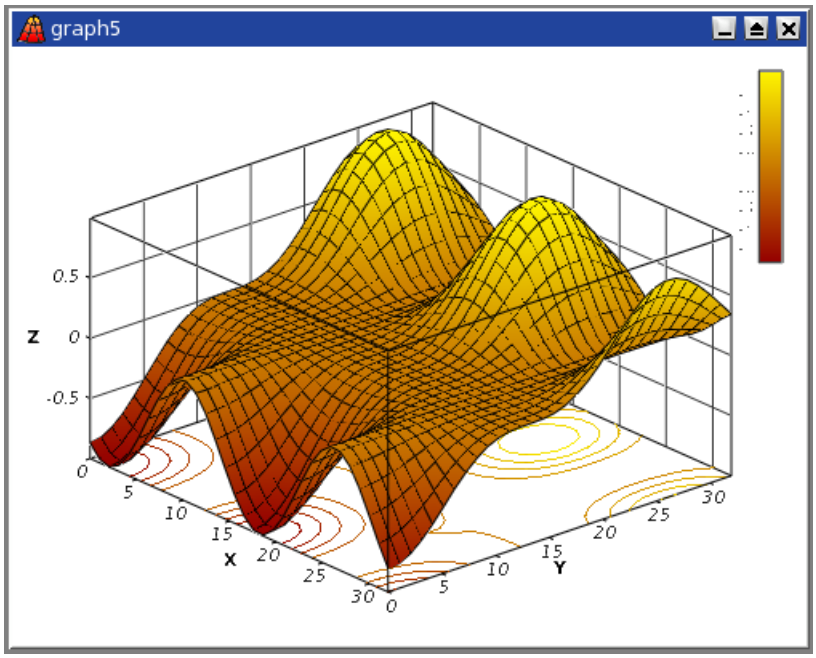


Figure 2.8: Example of a 3D Plots.

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 Plot 3D menu

2.2.1 Direct 3D plot from a function

This is the simplest way to obtain a 3d plot. It is done with the New -> New Surface 3D Plot command from the File menu or directly with the Ctrl-Alt-Z. This will open the folowing dialog box:

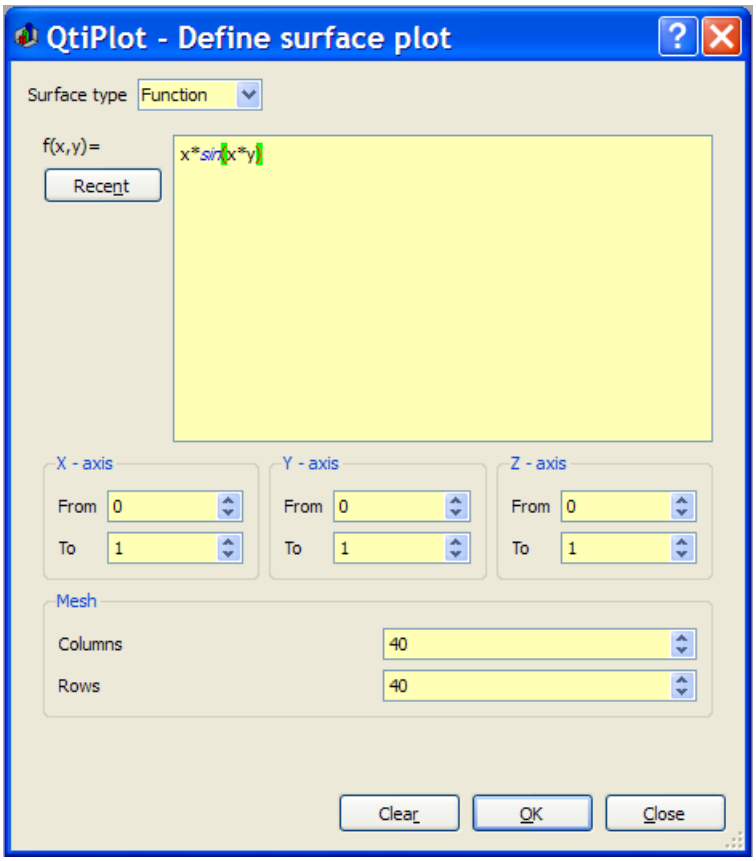


Figure 2.9: Definition of a new surface 3D plot

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

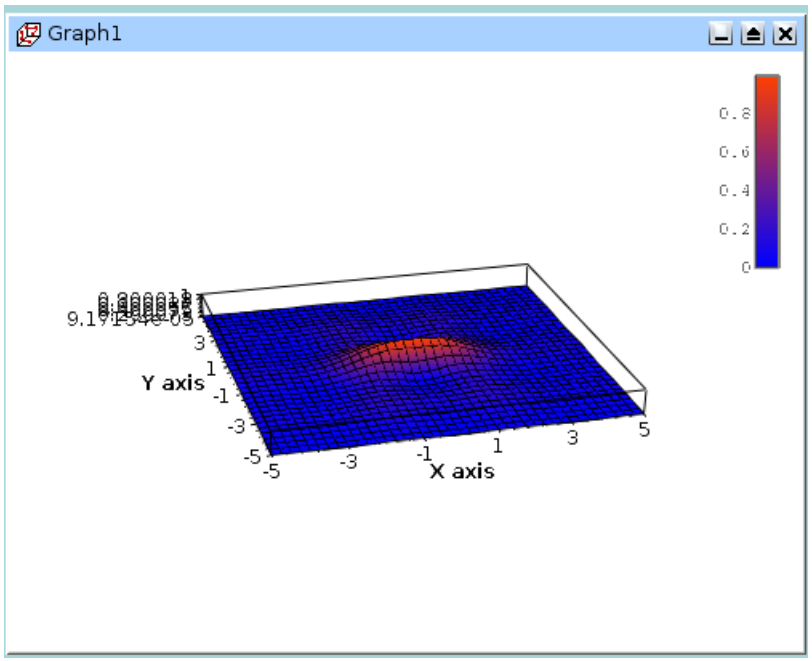


Figure 2.10: The 3D surface plot created by default

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 plot toolbar to add grids on the walls or to modify the style of the plot. After some modifications, you can obtain the following plot:

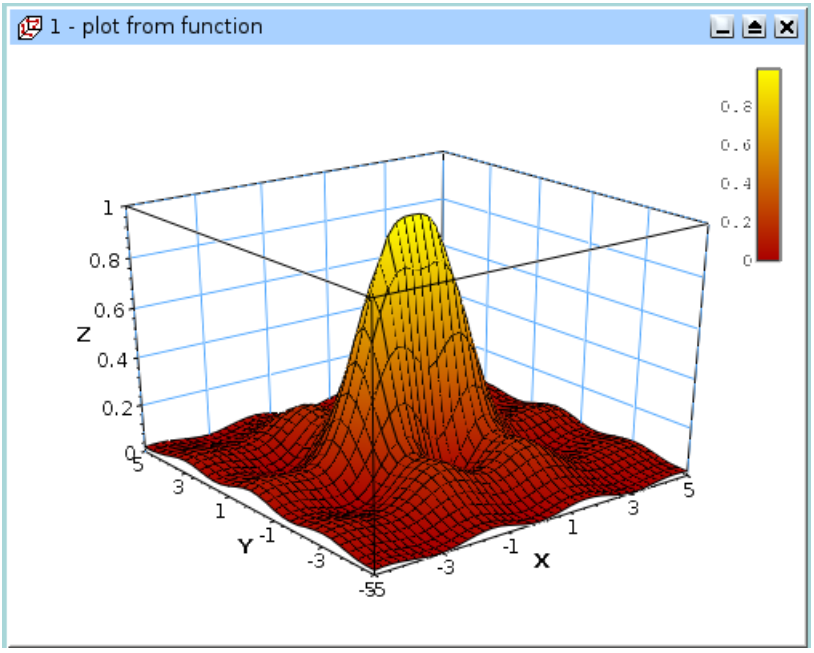


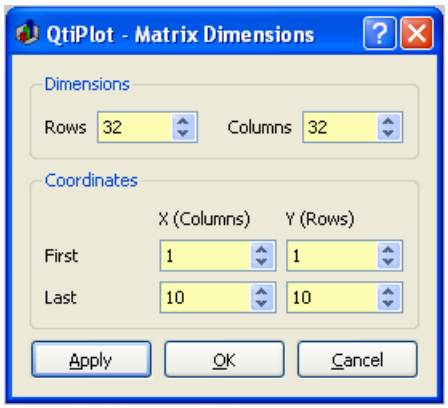
Figure 2.11: The 3D surface plot after customizations

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.

2.2.2 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 Set Dimensions... command to modify the number of rows and columns of the matrix. This dialog box is also used to define the X and Y ranges.



Then use the Set Values... 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 (see the section set-values for details).

The other way to obtain a matrix is to import an ASCII file into a table with the Import -> 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.

2.3 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 dialog. 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.

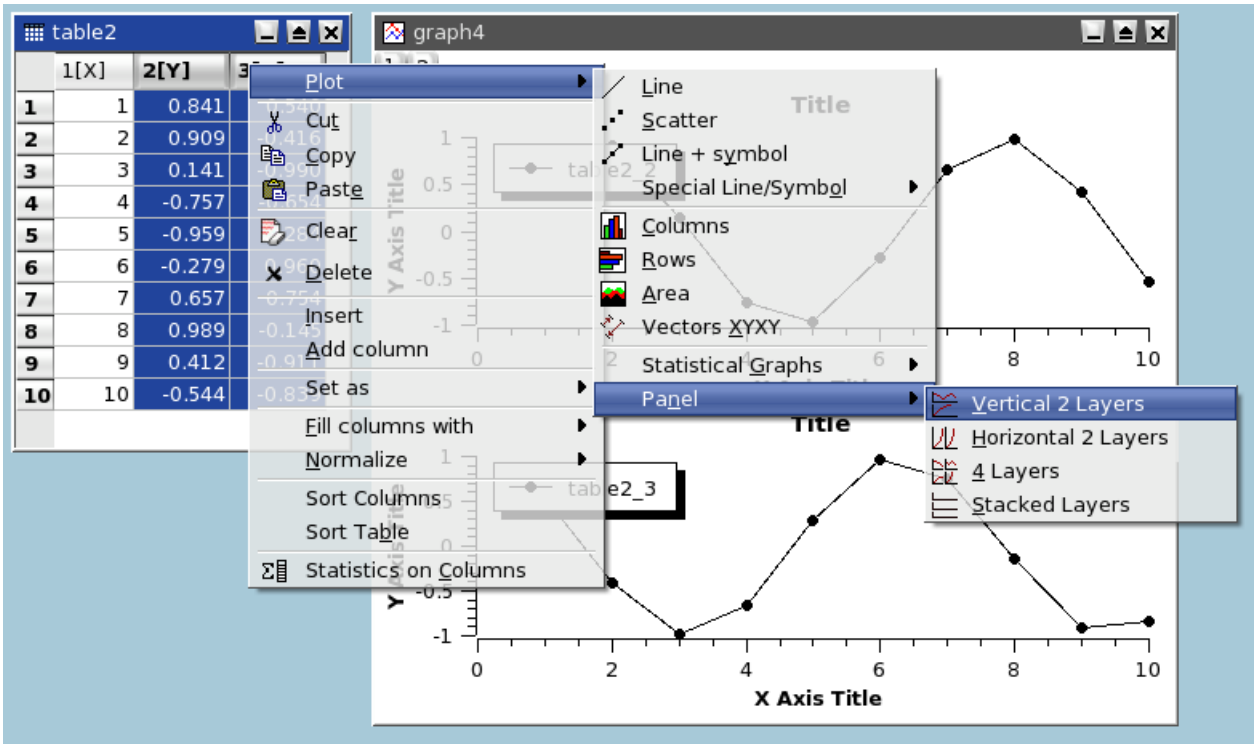
You can also use the mouse wheel in order to resize the layers: keeping the Ctrl key pressed and scrolling will resize the hight of the plot canvas, while keeping the Alt

key pressed and scrolling will resize its width. By keeping the Shift key pressed and scrolling you can resize the plot in both dimensions.

2.3.1 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. QtiPlot will create a panel of plots in which the size of the different elements of each plot are synchronized.

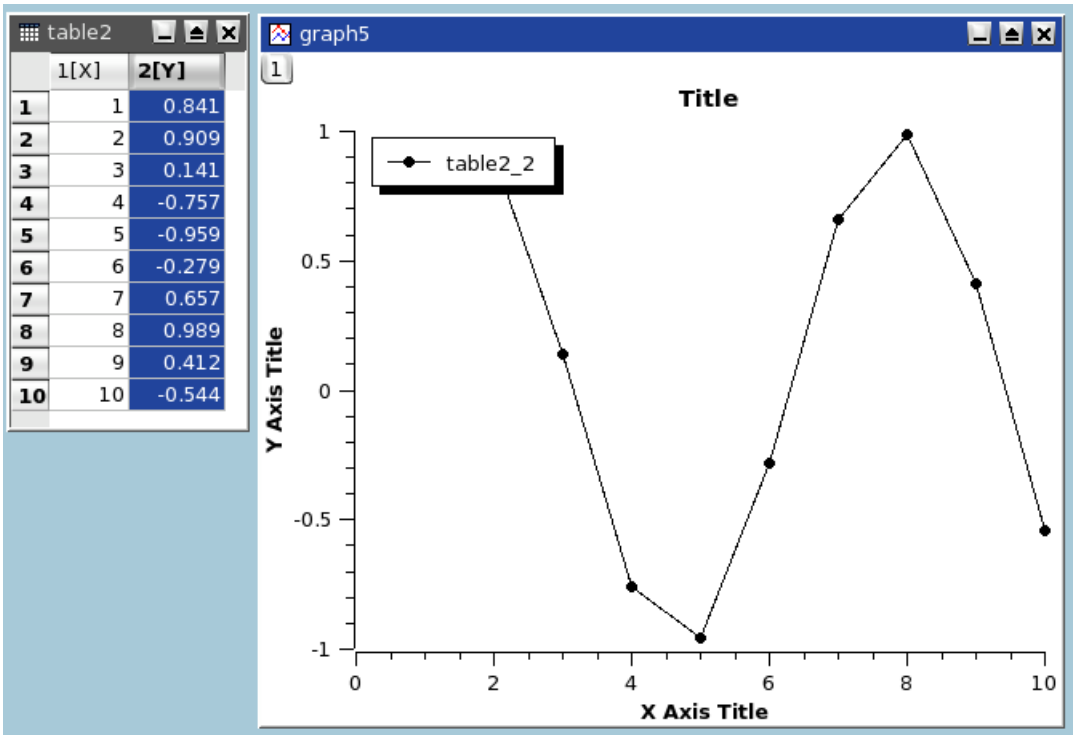


You can then customize the two plots, if you want to change the arrangement of the panel, you can use the Arrange Layers command from the Graph menu. It must be reminded in this case that each plot is in a layer with a surface which is the half or the quarter of the window surface area. So, if you want 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).

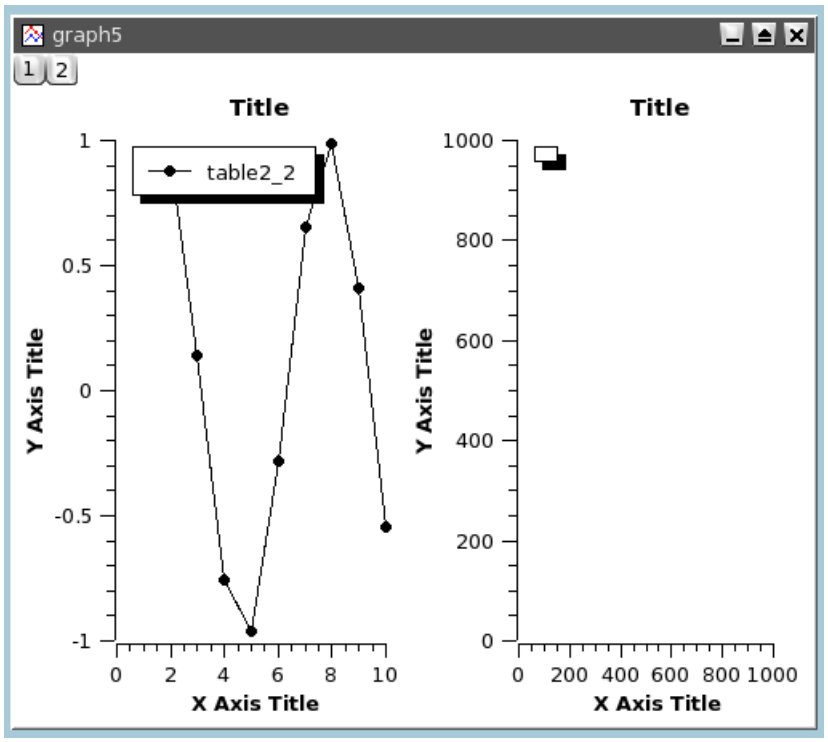
2.3.2 Building a multilayer plot step by step

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

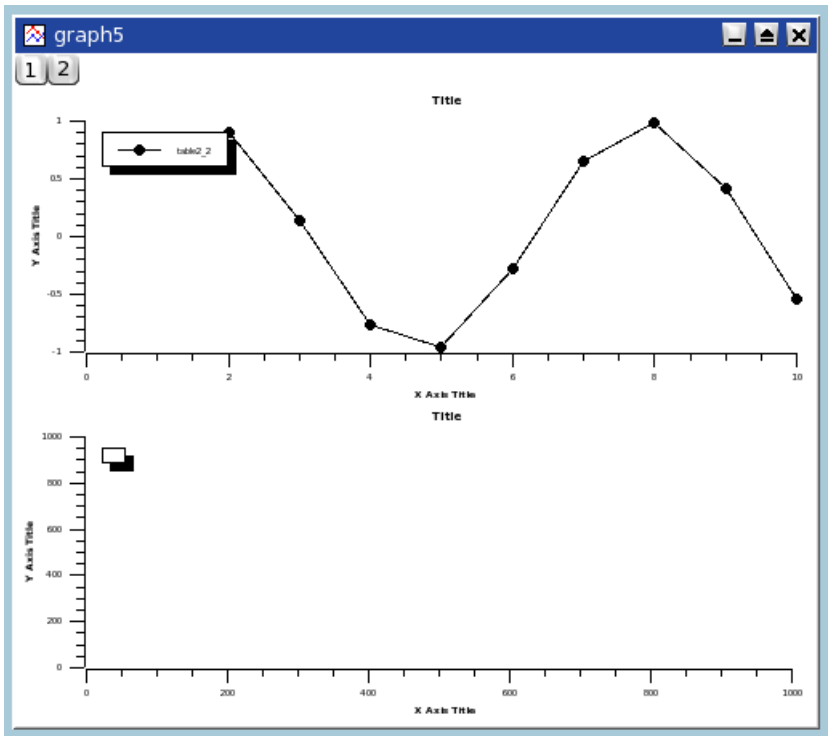
The first step is to build your first plot, for example from two columns of a table. We obtain a standard plot window:



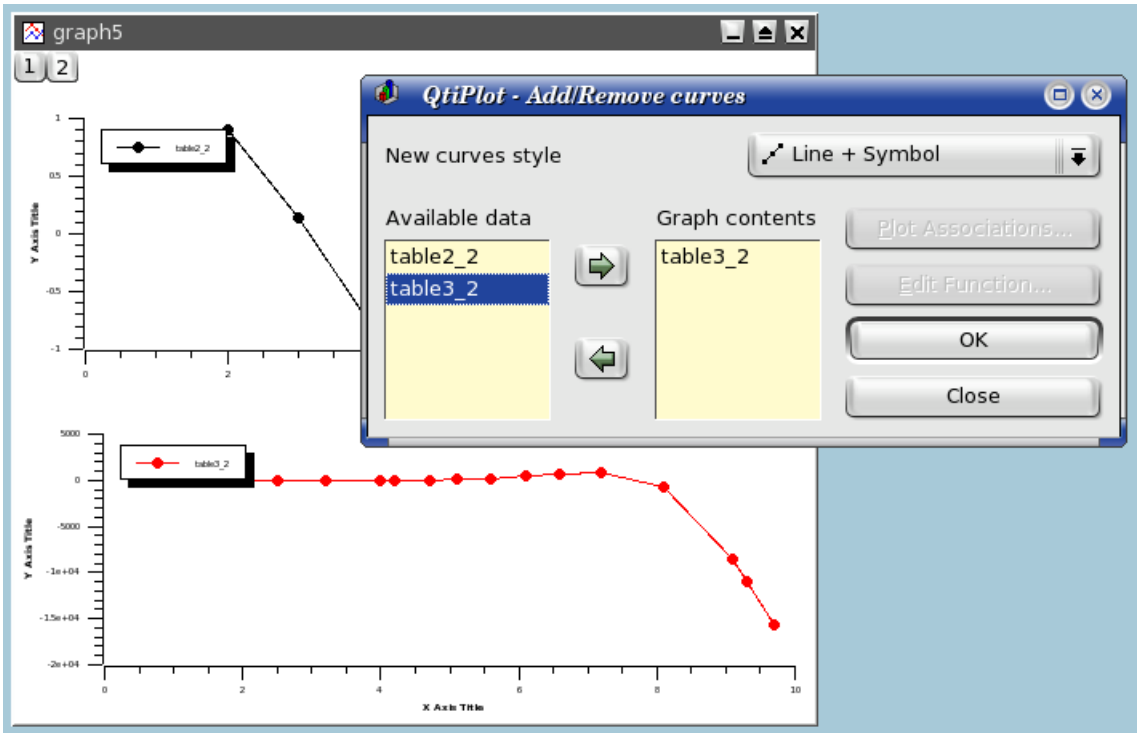
Then, select the plot window and use the Add Layer command from the Graph menu. This will activate the Add Layer dialog. If you choose "Guess" you will obtain a panel with two columns, if you choose "corner" you will obtain two superposed layers, you can then modify these two layers.



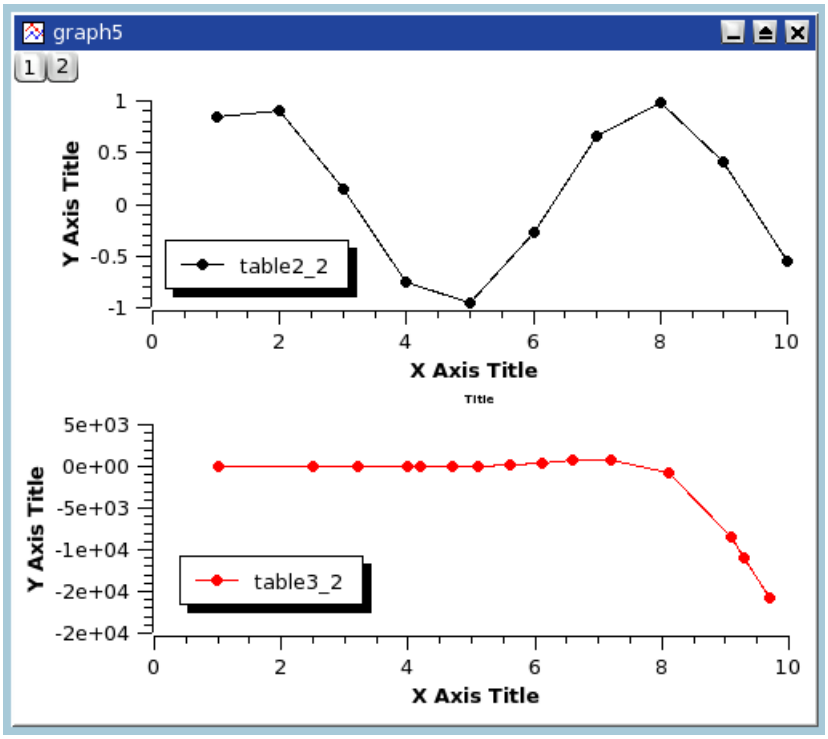
If you want to build a panel with two rows, you can use the Arrange Layers command to convert this panel.



Then select the second void plot and use the Add/Remove Curves... command to select the 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.


graph5
1
2
Y Axis Title
1
0.5
0
-0.5
-1
table2_2
0
2
4
6
8
10
X Axis Title
Title
Y Axis Title
5e+03
0e+00
-5e+03
-1e+04
-2e+04
-2e+04
table3_2
0
2
4
6
8
10
X Axis Title

Chapter 3

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.

3.1 The File Menu

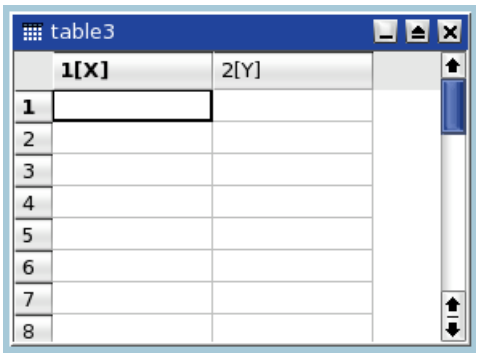
These commands can also be done by clicking on the *New Project* icon from the File Toolbar

File-> New ->

New -> New Project (Ctrl-N) Creates a new QtiPlot project file. 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.

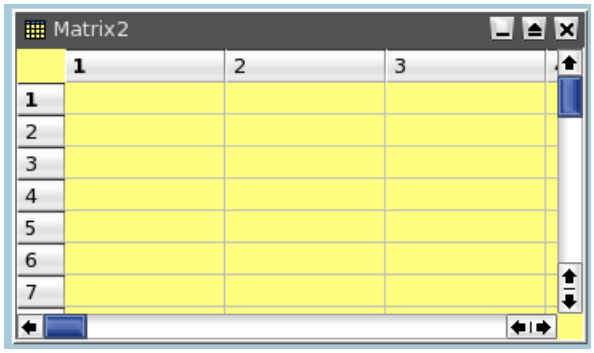
New -> New Folder (F7) Adds a new folder to the project. The new folder is added to the current folder.

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 Rows command and Columns command of the Table menu.



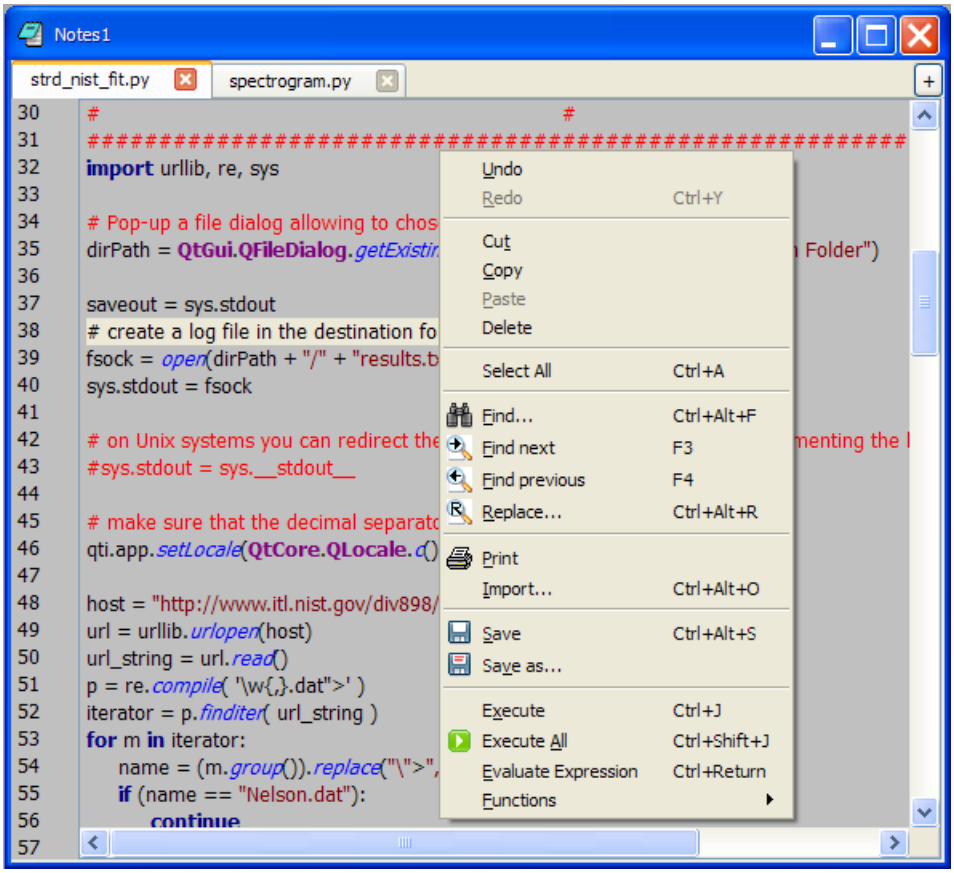
The properties of each column (format of numbers, width, etc) can be modified by the Column Options... command of the Table menu. See the table section for more details.

New -> New Matrix Creates a new Matrix into the project. The empty matrix will have 32x32 cells, these dimensions can be changed by the Set Dimensions... command of the Matrix menu

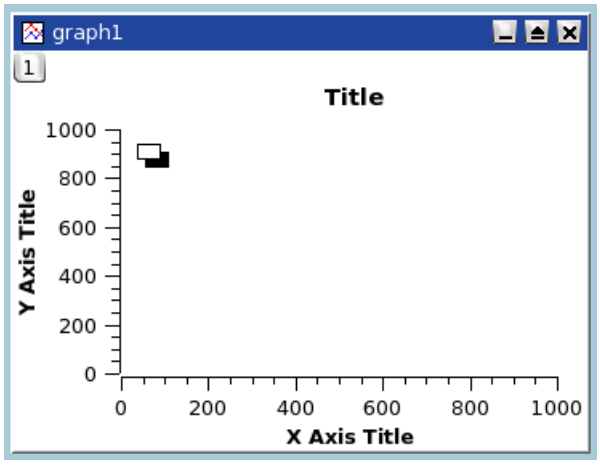


See the matrix section for more details.

New -> New Note 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.



New -> New Graph Creates a new empty 2D plot in the project. This default graph is just a framework in which you can add curves with the Add/Remove Curves... command.



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.

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

New -> New Surface 3D Plot (Ctrl-Alt-Z) Opens a dialog allowing to create a 3D plot by specifying an analytical function. Only cartesian coordinates are availables. See the 3D plot section of the tutorial for more detail on this function.

File -> Open (Ctrl-O) Opens an existing QtiPlot project file (.qti). If your project has been save in a compressed format, you must select the *.qti.gz* file format.

This command can also be used to open projects which have been built with the *Origin* software.

File -> Append Project... (Ctrl-Alt-A) Appends an existing QtiPlot project file (.qti) to the current project as a new folder.

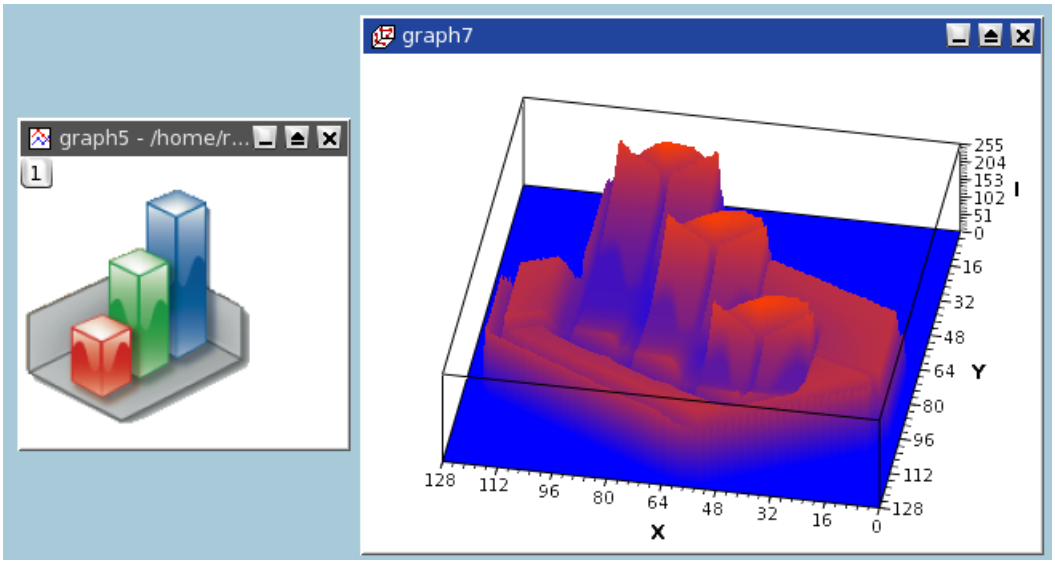
This command can also be used to open projects which have been built with the *Origin* software.

File-> Recent Projects Opens a list of the most recently used QtiPlot 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 -> Close Closes current project, without quitting the application.

File-> Open Image File This command loads an image file in a QtiPlot 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).

File-> Import Image... With this command, an image is loaded in the QtiPlot 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 QtiPlot logo.

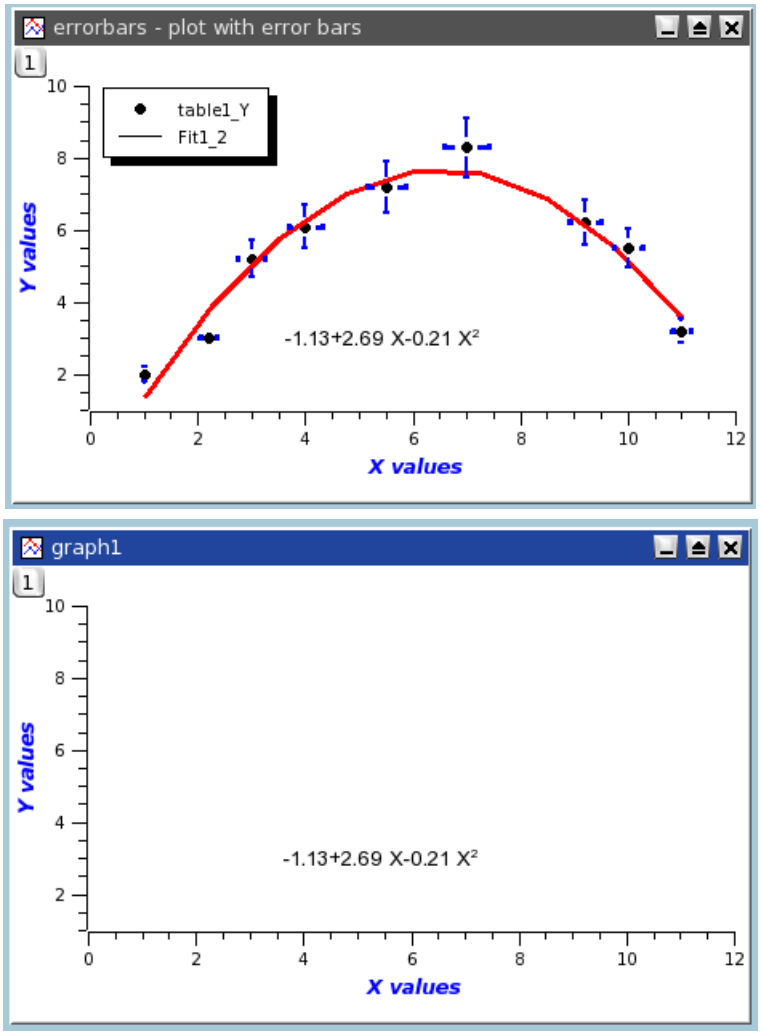
File-> Save Project (Ctrl-S) Saves the actual project. If the project hasn't been saved yet ("untitled" project), a dialog will open, allowing to save the project to a specific location.In a project file all settings and all plots are stored in ASCII format.

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 (.qti.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 QtiPlot plot file (.qpt). This command will create a new empty plot with the same graphical parameters (window geometry, fonts, colors, etc).

The first figure is the initial plot saved as a template, and the second one is the empty plot created by the **Open Template**.



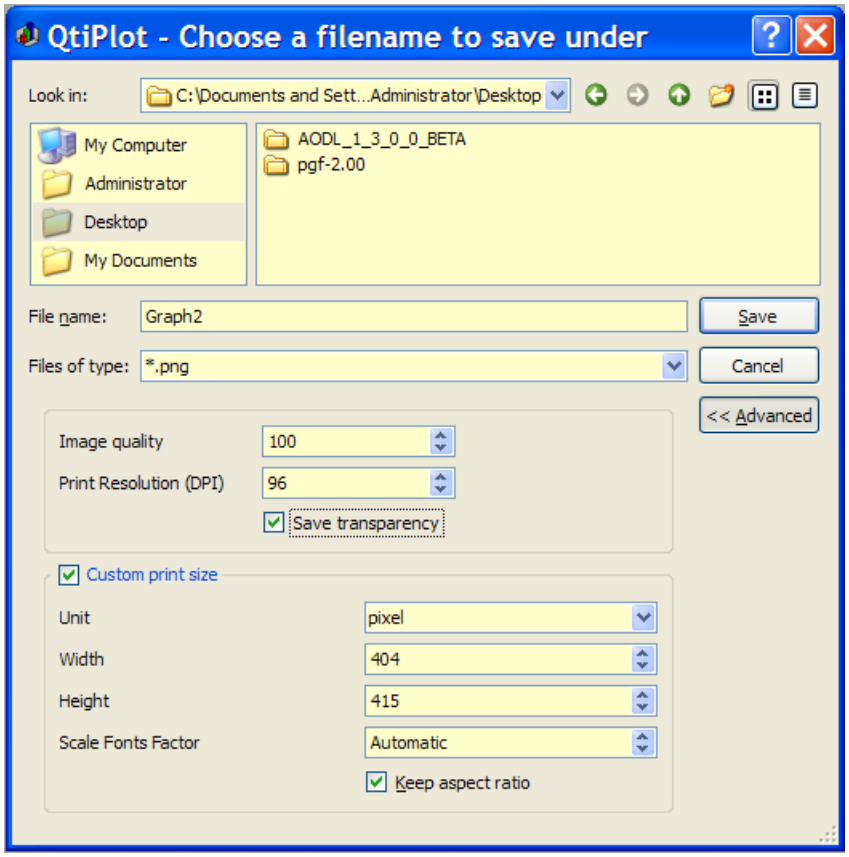
You just have to add curves with the Add/Remove Curves... command, but the style used to draw the curves is not kept in the template.

File -> Save as Template Save the active plot as a QtiPlot template file (.qpt). In this template, 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 *show options* check box. Depending on the chosen image format, the available options are not the same.

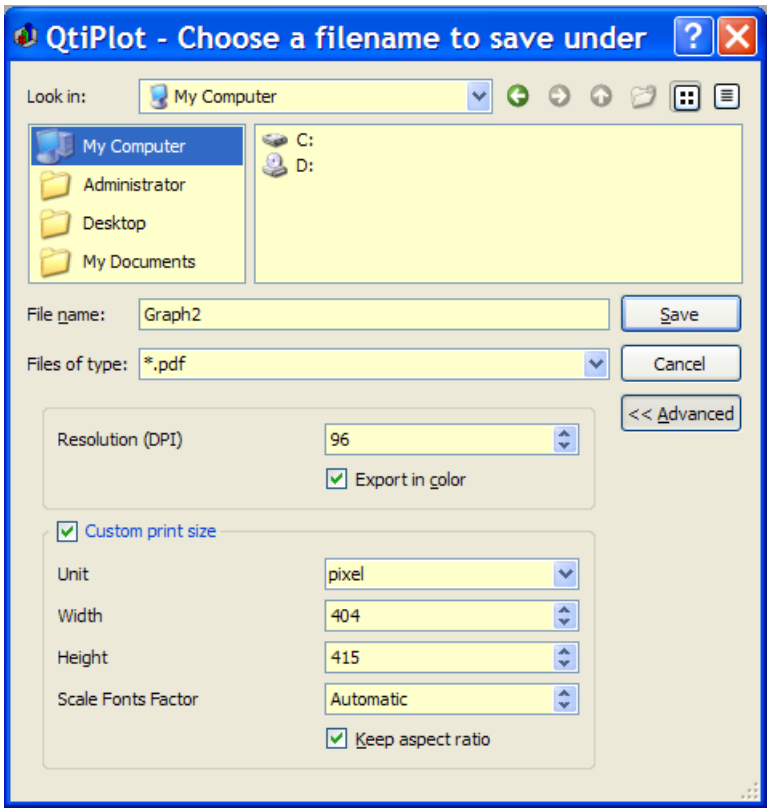
For *bmp, pbm, jpeg, xbm, pgm, ppm* image formats, the only option is the quality of the image, this parameter between 0 and 100% defines the compression ratio.

The higher it is, the best the quality is but the larger the file is. For *png*, *tiff* and *xpm*, you can choose to use a transparent background.

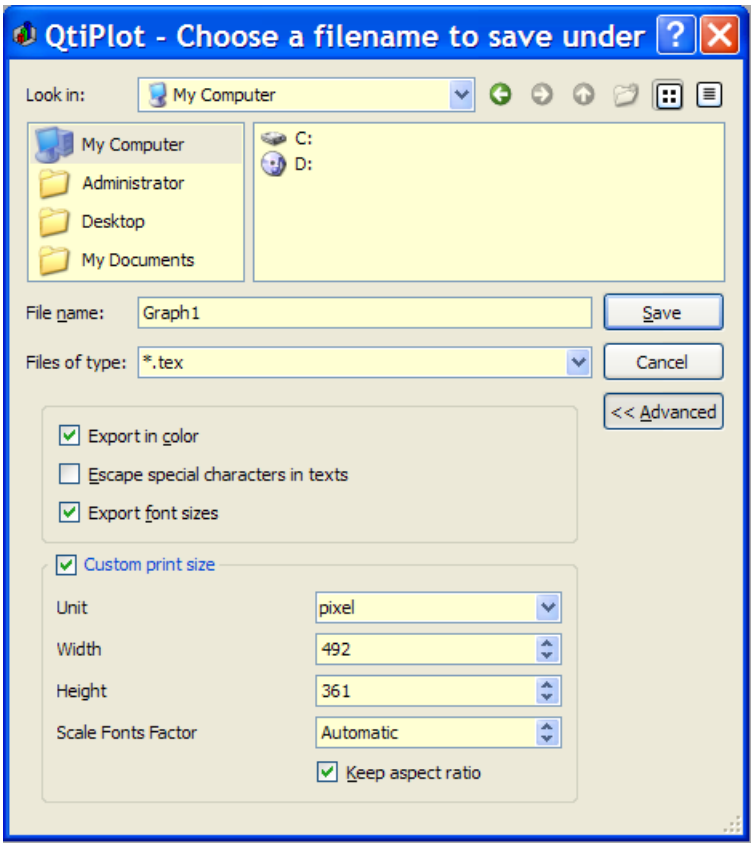


For *eps*, *pdf*, *ps*file formats, the option dialog is different. The main parameters availables are: the resolution and the size of the paper which is used to draw the plot. of the paper sheet. The default value for the resolution is the screen resolution. If you increase this parameter, the quality of the graphic elements will be better (but the overall size of the plot will be unchanged).

By default the plot is exported to its real size on screen, but if you want you can choose a different size, by checking the *Custom print size* box. In addition, you have the option to *Keep aspect ratio* of the plot. If you check this box and you modify one dimension of the plot, the other dimension will be automatically modified so that the plot aspect remains the same.



When exporting the plot to LaTeX (.tex) you have two very useful options: *Export font sizes* and *Escape special characters in title/axis labels*. If checked, the first option allows you to keep the original font sizes. If not, the font size specified in the preamble of the TeX document will be used for all text strings in the plot. The second option specifies if LaTeX special characters should be escaped or not when exporting. If the title or the axis labels contain LaTeX syntax (like superscripts, subscripts, etc...) and you want them to be interpreted by the LaTeX compiler, you should uncheck this option.



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.

File -> Create Open Document Presentation... Here you have the possibility to save all plots of the project to an Open Document Format file (.odf) that can be opened and edited with OpenOffice.

File-> Print (Ctrl-P) Prints the active plot. A print dialog is opened where you can select the printer, different paper sizes, etc.

File-> Print Preview Displays a print preview for the active window. You can also use this dialog in order to actually print the window.

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 the Export ASCII dialog allowing to save the data from the active spreadsheet to an ASCII file.

File -> Import -> Import ASCII... (Ctrl-K) The options for the importation of ASCII data files are set by the Import dialog.

File -> Import -> Sound (WAV)... This allows you to import an uncompressed sound .wav file (PCM format).

File -> Quit (Alt-F4) Closes the application. You will be asked wether you want to save your last changes or not.

3.2 The Edit Menu

Edit -> Undo (Ctrl-Z) Restores the last modified table at the state it had after the last "Save Project" operation. Not available for plot windows.

Edit -> Redo (Ctrl-R) Restores the modifications in a table after a "Undo" operation. Function not available for plot windows.

Edit -> Cut Selection (Ctrl-X) Copies the current selection to the clipboard and deletes the selection. It currently works for spreadsheets and for 2D plots objects.

Edit -> Copy Selection (Ctrl-C) Copies the current selection to the clipboard. It currently works for spreadsheets and for 2D plots objects.

Edit -> Paste Selection (Ctrl-V) Pastes the content of the clipboard to the active window.

Edit -> Delete Selection () Cleares the current selection. It 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 Informations Deletes from the project file all the history information about the analysis operations performed by the user. The log panel is then empty.

View -> Preferences... Opens the Preferences dialog.

3.3 The View Menu

View -> Toolbars... (Ctrl-Shift-T) Opens a pop-up menu allowing to enable/disable tool bars.

View -> Plot Wizard (Ctrl-Alt-W) Opens the Plot Wizard dialog.

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.

View -> Results log Opens/Close a panel displaying the historic of the data analysis operations performed by the user.

3.4 The Graph Menu

This menu is only active when a plot window is selected.

Graph -> Add/Remove Curves... (Alt-C) Opens the Add/Remove Curves... 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.

Graph -> Add Error Bars... (Ctrl-B) Opens the Add Error Bars... dialog. You can add error bars on X and/or on Y values on an existing plot.

Graph -> Add Function... (Ctrl-Alt-F) Opens the Add Function... dialog. This command allows to add a a new curve on an existing plot.

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 -> Add Text (Alt-T) 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...)

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.

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 -> Add Layer (Alt-L) Opens a dialog allowing you to select whether the new layer is to be added to the left-top corner of the plot window or to a best-guess position (based on a layer positioning algorithm in columns and rows).

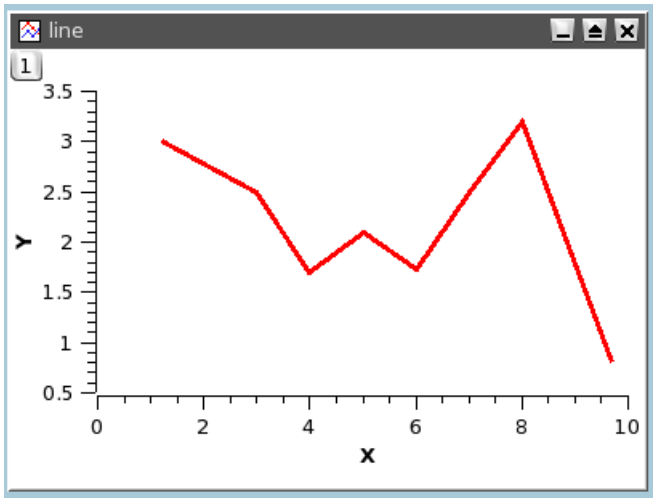
Graph -> Remove Layer (Alt-R) Deletes the active layer and prompts out a question dialog allowing you to choose whether the remaining layers should be automatically re-arranged or not.

Graph -> Arrange Layers (Shift-A) Opens the Arrange layers dialog, allowing you to custom the layout of the active 2D plot window.

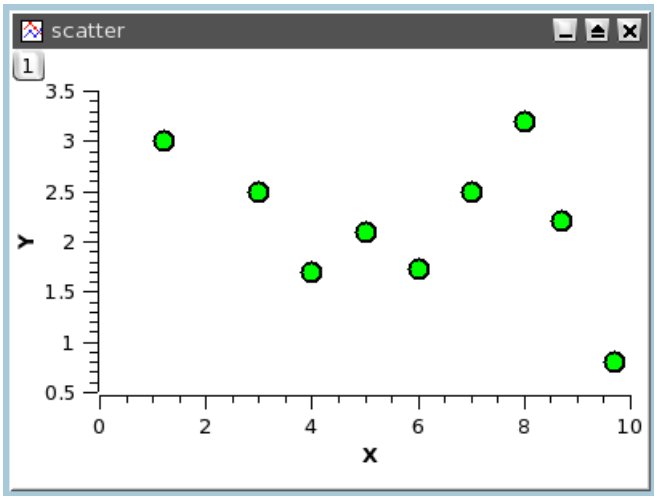
3.5 The Plot Menu

This menu is active only when a table is selected. These commands allow to plot the data selected in the active table.

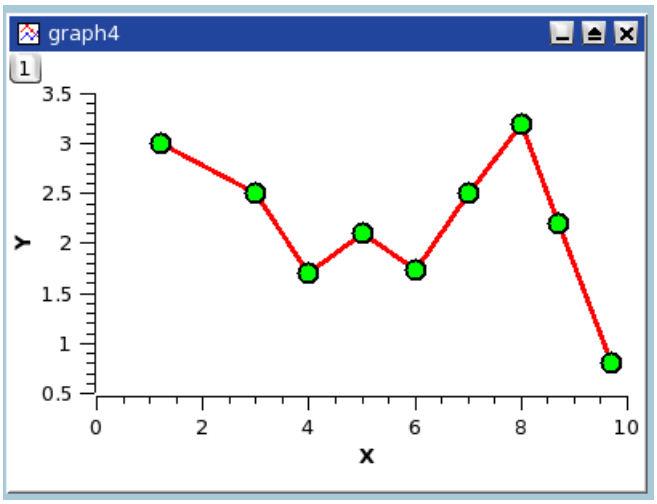
Line Plots the selected data columns in the active table window using the "Line" style. This command can also be activated by clinking on the ⁄ icon of the Table toolbar. Once the plot is created, the drawing of the data series can be customized with the Custom curves dialog.



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 with the Custom curves dialog.

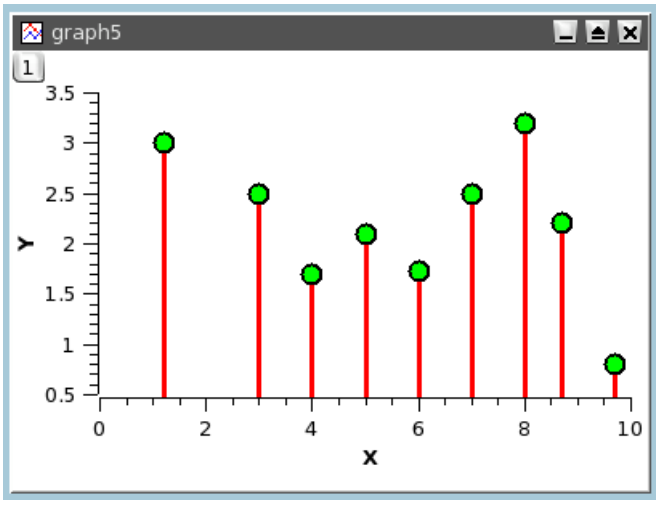


Line+Symbol Plots the selected data columns in the active table window using the "Line + Symbol" style.This command can also be activated by clinking on the ⋰ icon of the Table toolbar. Once the plot is created, the drawing of the data series can be customized with the Custom curves dialog.

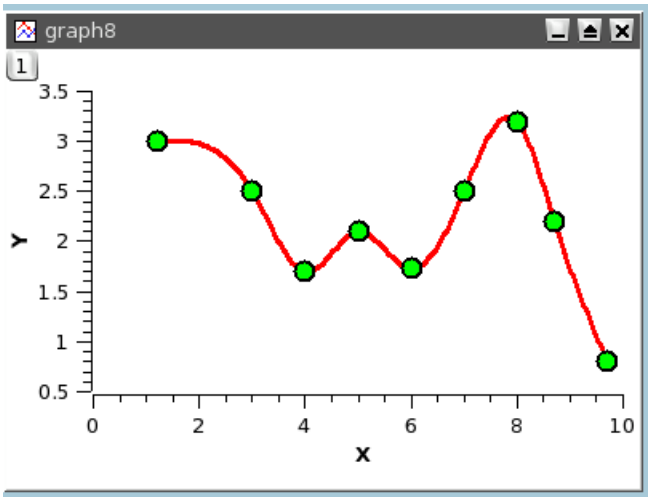


Special Line+Symbol -> Vertical Drop Lines Plots the selected data columns in the active table window using the "Vertical drop lines" style. Once the plot is

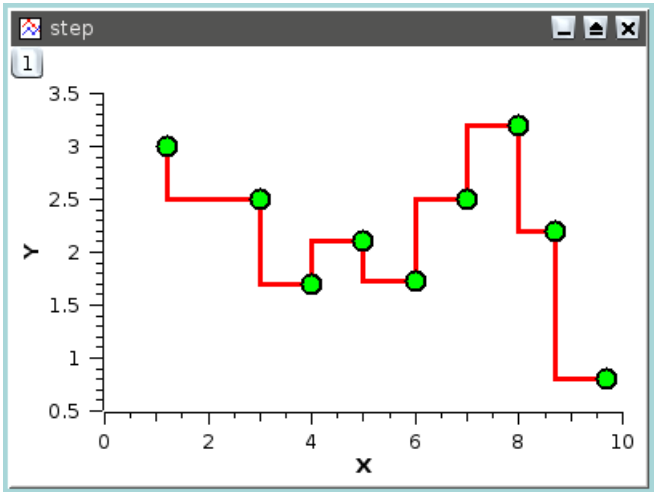
created, the drawing of the data series can be customized with the Custom curves dialog.



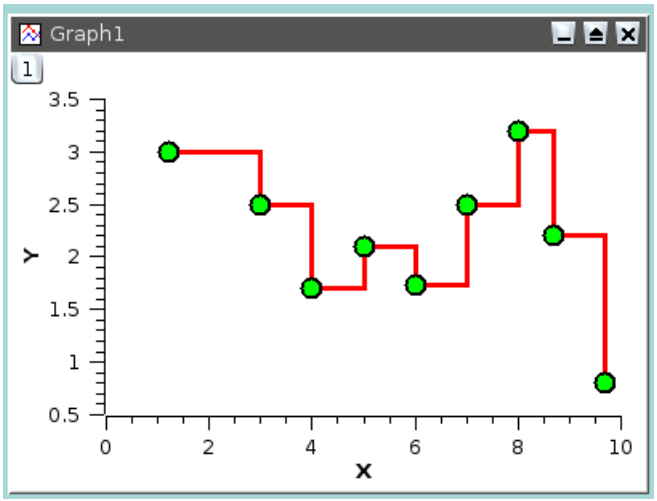
Spline Plots the selected data columns in the active table window using the "Spline" style. Once the plot is created, the drawing of the data series can be customized with the Custom curves dialog.



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 with the Custom curves dialog.

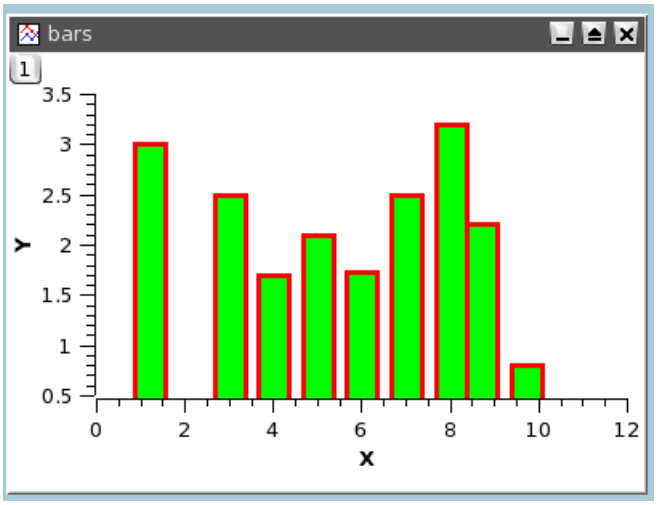


Horizontal Steps Plots the selected data columns in the active table window using the "Horizontal Steps" style. Once the plot is created, the drawing of the data series can be customized with the Custom curves dialog.

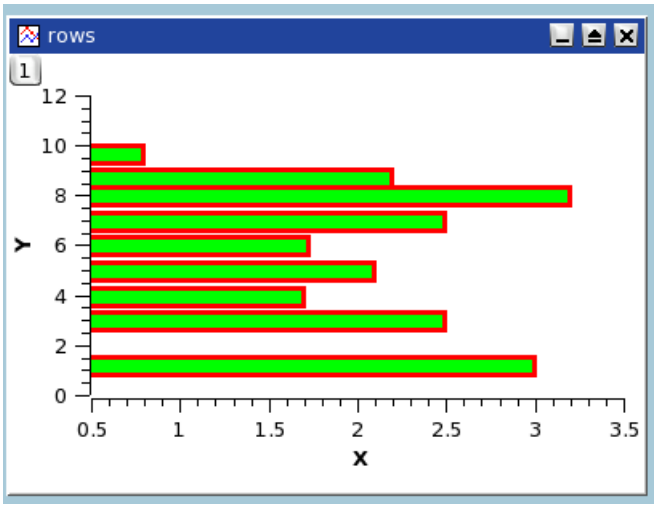


Double-Y Creates a double Y axis graph. Requires a selection of at least two Y columns (or a range from at least two columns).

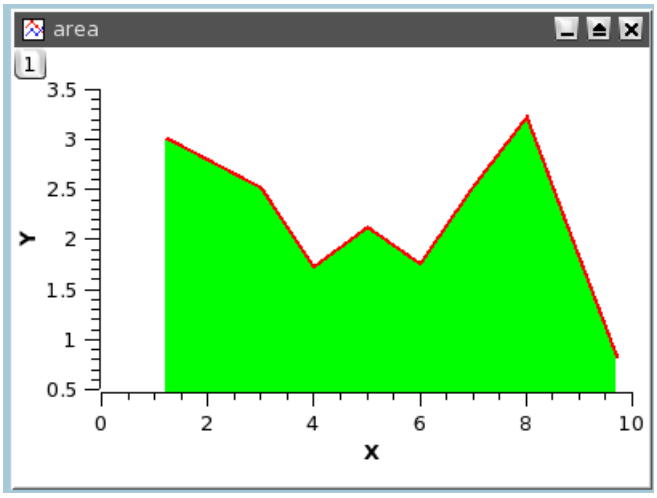
Columns Plots the selected data columns in the active table window using the "Columns" style, that is vertical bars.



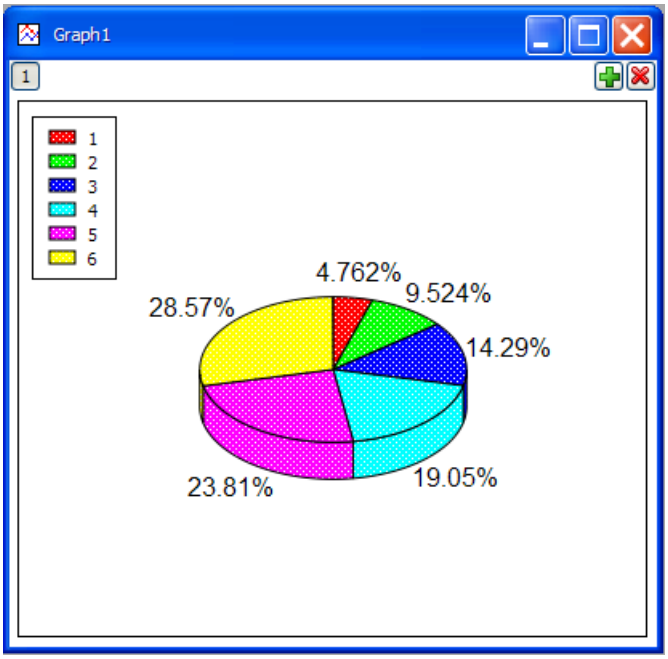
Rows Plots the selected data columns in the active table window using the "Rows" style.



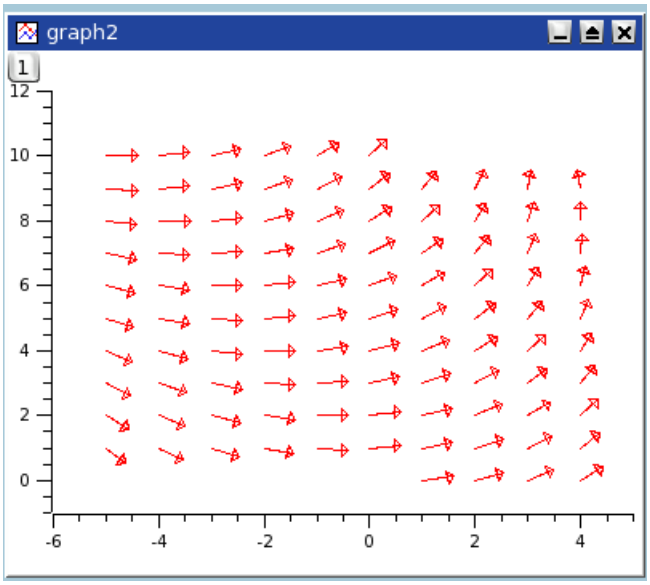
Area Plots the selected data columns in the active table window using the "Area" style.



Pie Creates a pseudo 3D Pie plot of the selected column in the active table window (only one column allowed).



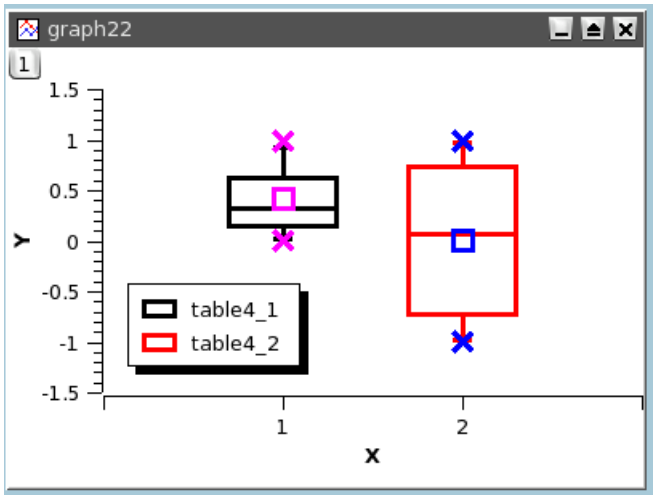
Vectors XYXY Creates a vectors plot of the selected column in the active table window. You must select four columns for this particular type of plot. The two first columns give the coordinates for the starting points of the vectors, the two last columns giving the information regarding the end points.



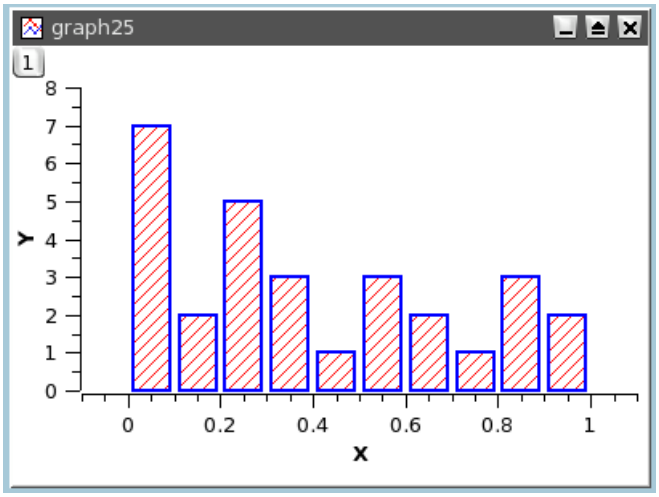
Vectors XYAM Creates a vectors plot of the selected column in the active table window. You must select four columns for this particular type of plot. The two first columns give the coordinates for the starting points of the vectors, the two last columns giving the angle (in radians) and the magnitude of the vectors.

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 with the Custom curves dialog.



Statistical Graphs -> Histogram Creates a frequency histograms of the selected data columns in the active table window. The default binning uses 10 steps between the max and the min of Y-values. This can be modified with the Custom curves dialog.



With this command, a frequency distribution is computed from your data. If you want to draw an histogram directly from values, use the **Bars**.

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.

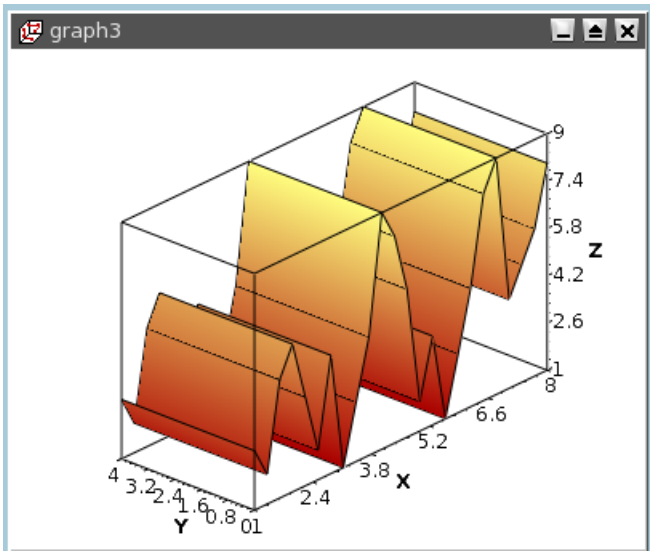
Panel -> Vertical 2 Layers Creates 2 vertically stacked layers displaying the selected data columns in the active table window (one curve per layer).

Panel -> Horizontal 2 Layers Creates 2 horizontally stacked layers displaying the selected data columns in the active table window (one curve per layer).

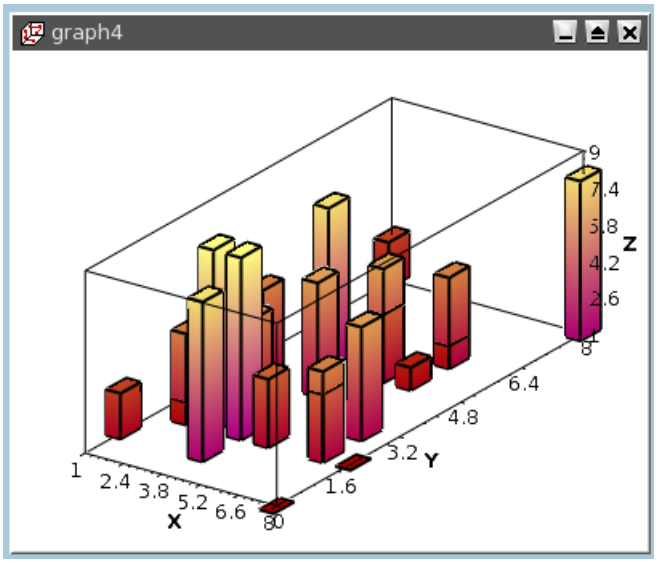
Panel -> 4 Layers Creates 4 layers on a 2x2 grid, displaying the selected data columns in the active table window (one curve per layer).

Panel -> Stacked Layers Creates vertically stacked layers displaying the selected data columns in the active table window (one curve per layer).

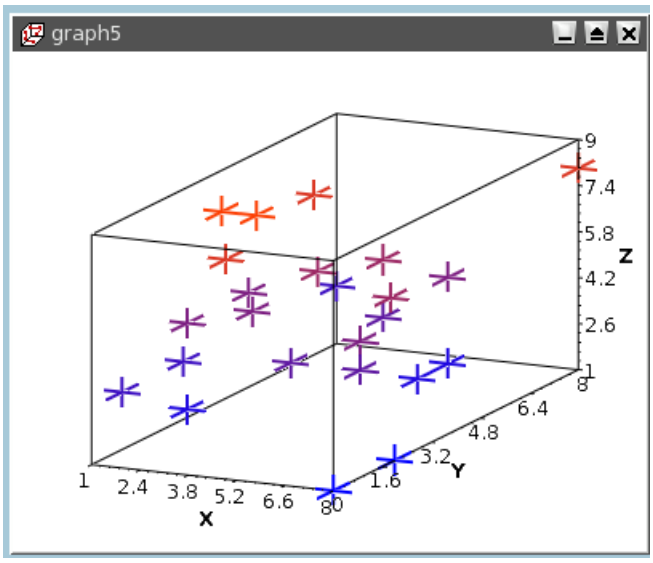
Data -> Plot 3D -> Plot 3D -> Ribbons Makes a 3D plot of the selected data column in the active table window (only one column allowed) using the "Ribbon" style.



Bars Makes a 3D plot of the selected data column in the active table window (only one column allowed) using the "3D Bars" style.



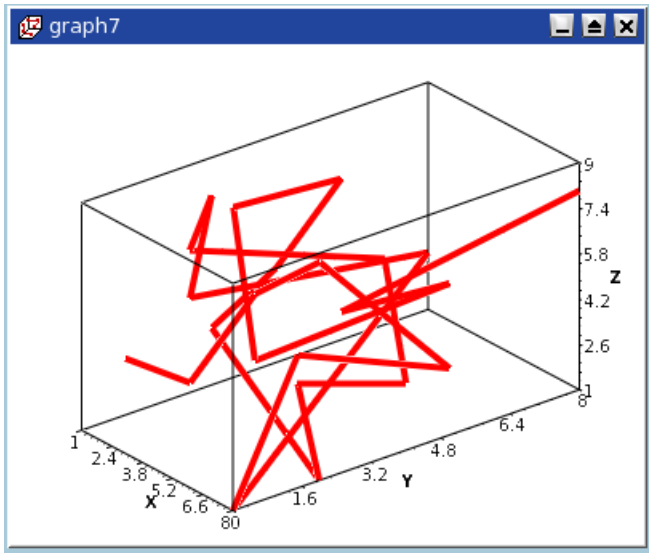
Scatter Makes a 3D plot of the selected data column in the active table window (only one column allowed) using the "3D Dots" style. The 3D point symbol style can be changed via the 3D Plots Settings dialog.



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 plot toolbar (respectively and for cross-hairs, dots and cones) or with the custom-curves dialog.

Plot 3D -> Trajectory Makes a 3D plot of the selected data column in the active

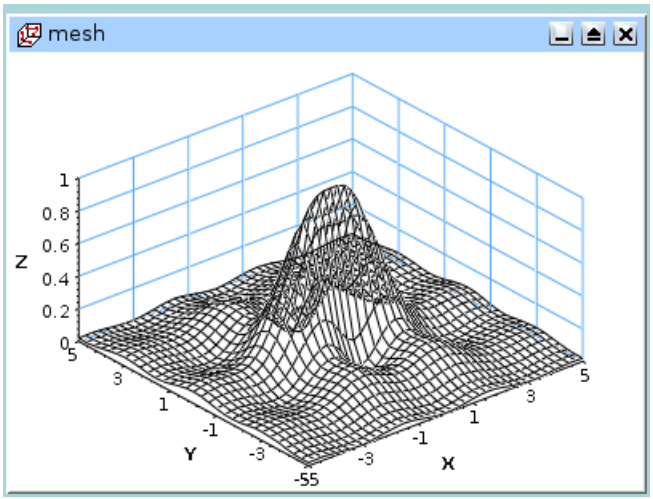
table window (only one column allowed) using the "3D Line" style. The line width and color may be changed via the 3D Plots Settings dialog.



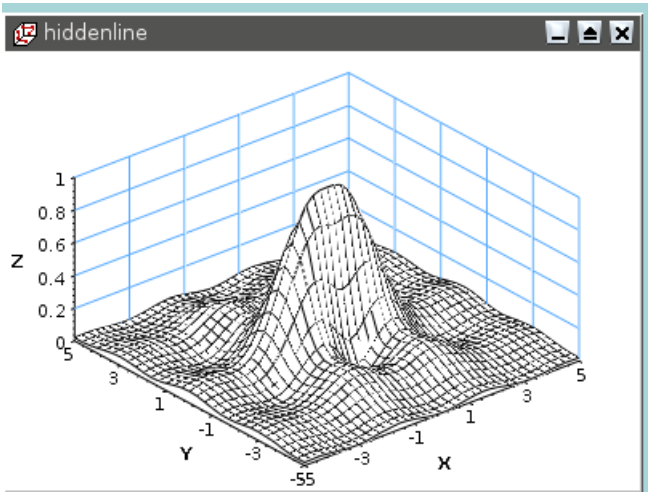
3.6 The Plot 3D menu

This menu is only active when a matrix is selected.

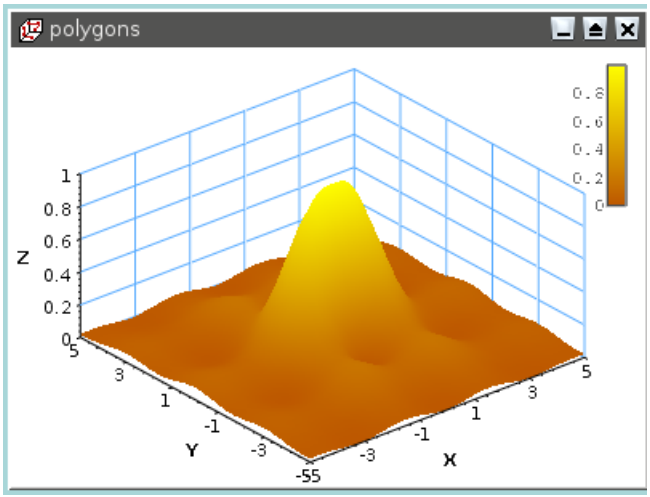
3D Wire Frame Makes a 3D plot of the selected matrix using the "3D mesh" style.



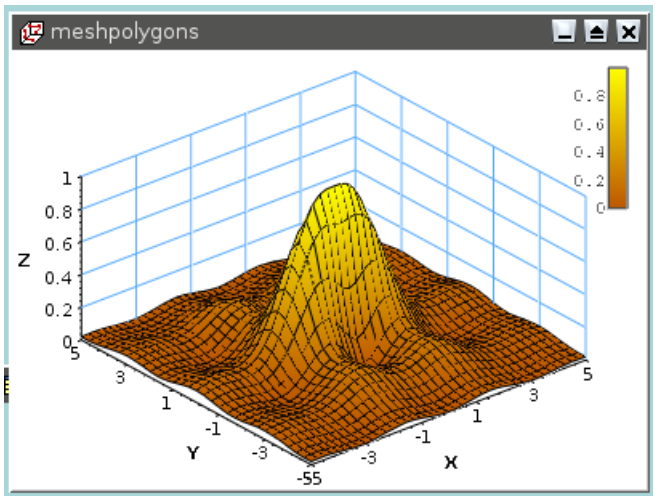
3D Hidden Lines Makes a 3D plot of the matrix using the "3D mesh" style with hidden lines.



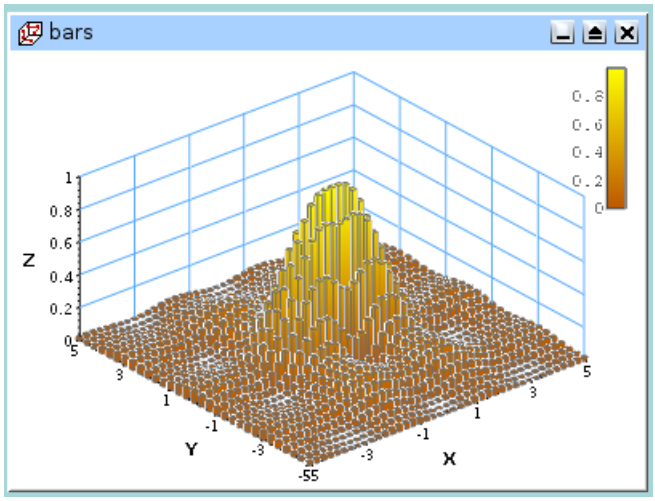
3D Polygons Makes a 3D plot of the matrix using the "3D polygons" style.



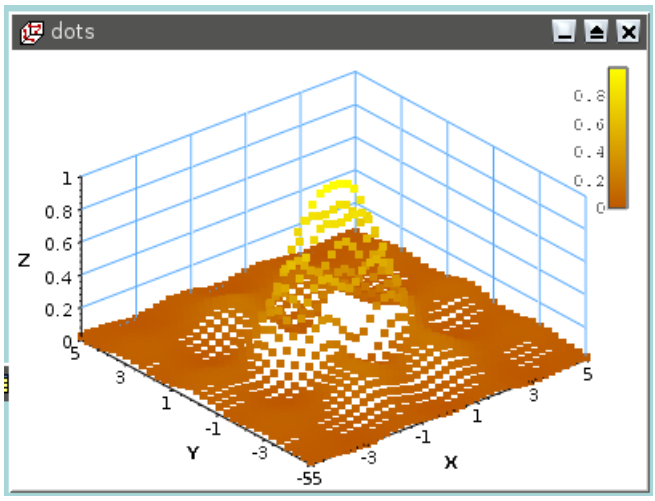
3D Wire Surface Makes a 3D plot of the matrix using the "3D polygons" style with the mesh drawn.



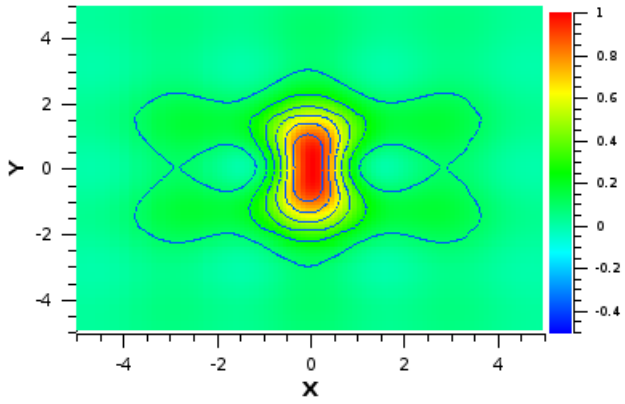
Bars Makes a 3D plot of the selected data column in the active table window (only one column allowed) using the "3D Bars" style.



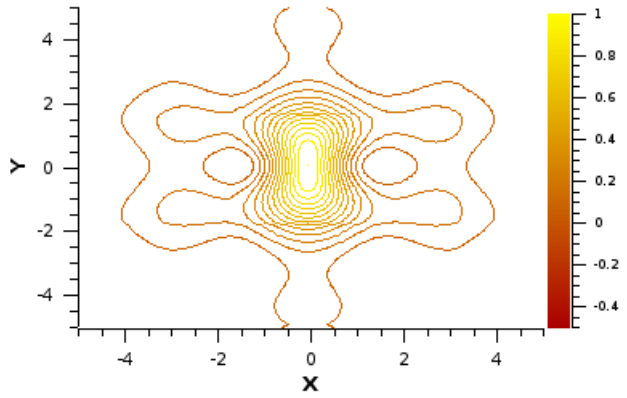
Scatter Makes a 3D plot of the selected data column in the active table window (only one column allowed) using the "3D Dots" style. The 3D point symbol style can be changed via the 3D Plots Settings dialog.



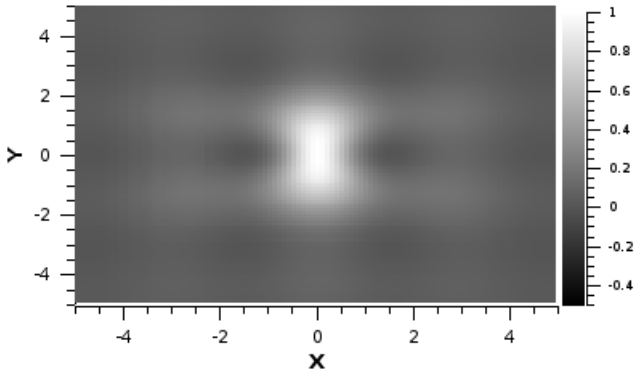
Contour+Color Fill Makes a color map plot of the data in the active matrix window. The contour lines and the colormap settings may be changed by clicking on the plotting area, this will active the Contour Options Dialog.



Countour 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.



3.7 The Data Menu

This menu is active only when a plot is selected.

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 after a zoom operation.

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. The [Home] key makes the data reader jump to the beginning of the selected curve and the [End] key to its end point.

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. When this tool is active you can easily copy, paste or cut the whole selected data range, using the well-known shortcuts: Ctrl+C, Ctrl+V and Ctrl+X, respectively.

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 -> 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.

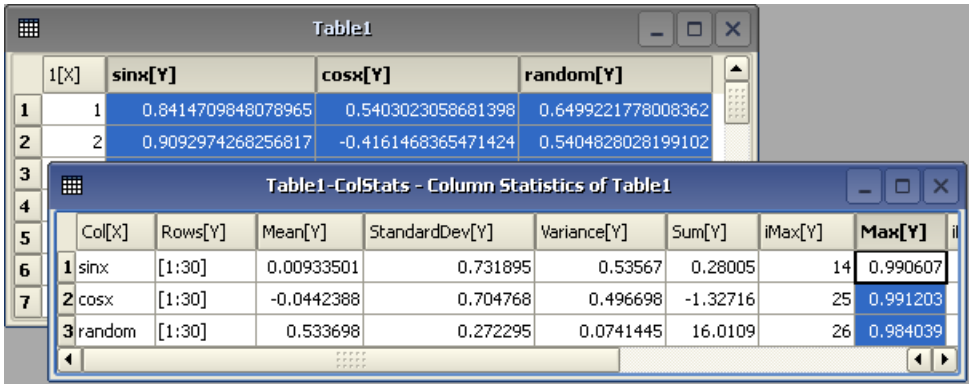
Data -> Remove Bad Data Points (Alt-B) Allows you to remove data points from the active plot layer by double-clicking on them. The coordinates of the points selected for removal are shown in the Data Display toolbar. The changes you make automatically modify the data into the corresponding tables and all the plots depending on those data sets.

3.8 The Analysis Menu

The commands which are available in this menu are not the same if a table or a plot is selected.

3.8.1 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...



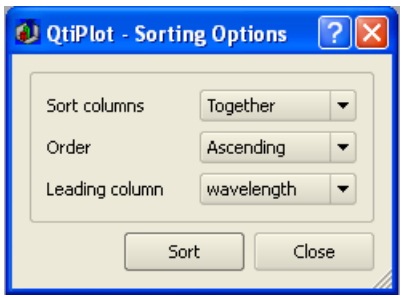
You can select several columns in one table, one line will be created for each column. You can't select columns in different tables to obtain one single table of statistics.

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.

Sort Column Sorts the columns selected. If more than one column is selected, you can sort them:

- separately: each column will be sorted in ascending or descending order
- together: the column selected as *leading column* will be sorted in ascending or descending order, and the others column selected will be sorted in order to keep the rows unchanged.



Sort Table This is the same command as Sort Column but it operate on all columns of the active table.

Normalize Normalizes the columns selected, that is modify the data in order to obtain a range of 0 to 1. All columns selected are normalized separately. This command doesn't create new normalized columns but replace the values of the selected columns.

Normalize -> Columns Normalizes the selected column.

Normalize -> Table Normalizes all the columns of the table, it is not a global normalization of all values of the table: each column is normalized separately.

FFT... Computes a direct or inverse Fast Fourier Transform. The parameters used can be set with the FFT dialog. See the fft section of the Analysis chapter for more details.

Correlate Does a cross-correlation of the two columns which are selected. See the correlate section of the Analysis chapter for more details.

Autocorrelate Does a cross-correlation of the selected column. See the correlate section of the Analysis chapter 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 convolution section of the Analysis chapter 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 deconvolution section of the Analysis chapter 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 Non Linear Curve Fit section of the Analysis chapter for more details.

3.8.2 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 display.

The commands FFT... and Fit Wizard... are presented in the Table Analysis Menu.

Analysis -> Differentiate Creates a new plot displaying the resulting curve of the numerical differentiation. The computation of the derivative is done by centered finite diferences.

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.

Analysis -> 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 result is given in the Log Panel.

Analysis -> Smooth .

Savitski-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.

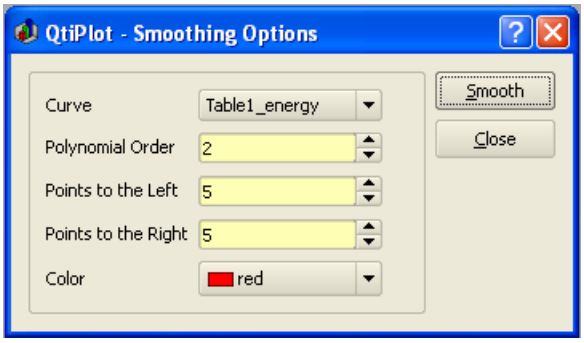


Figure 3.1: The **Smooth -> Savitsky-Golay...** dialog.

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.

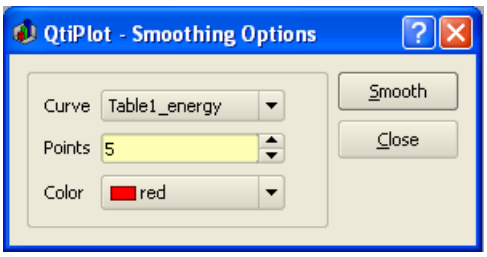


Figure 3.2: The **Smooth -> Moving Window Average...** dialog.

Lowess... This command performs a smoothing of the selected curve using the Lowess (aka Loess) algorithm. It provides a robust locally weighted regression and is well suited to smooth data for which no formal model exists.

It's parameter *f* is the fraction of points which define the local neighborhood. A value of 0.2 uses 20% of the curve total points as neighbors for each data point (+/- 10%). Choose it closer to 1 for a smoother curve. The parameter *iterations* tells how often the algorithm runs over the whole data, each time refining the local weights. For most cases two iterations are enough.

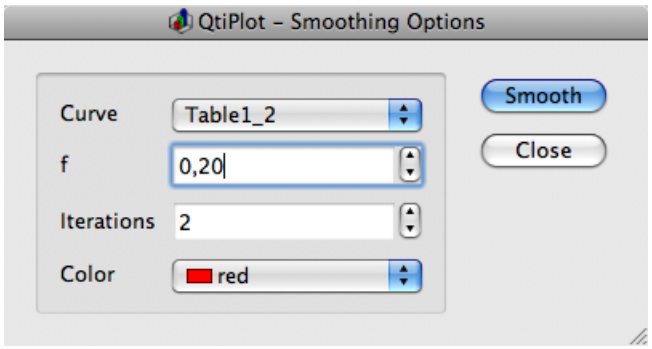


Figure 3.3: The **Smooth -> Lowess...** dialog.

Analysis -> 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.

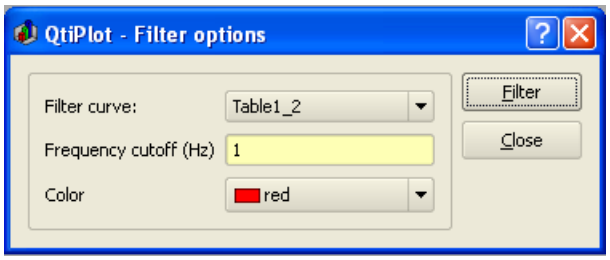


Figure 3.4: The **FFT Filter -> Low Pass...** dialog.

This command creates a new table with the filtered data, and a new curve will be added on the current plot.

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.

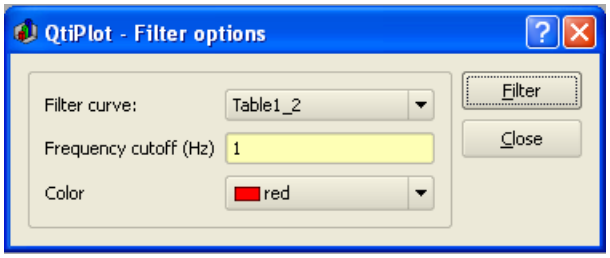


Figure 3.5: The **FFT Filter -> High Pass...** dialog.

This command creates a new table with the filtered data, and a new curve will be added on the current plot.

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.

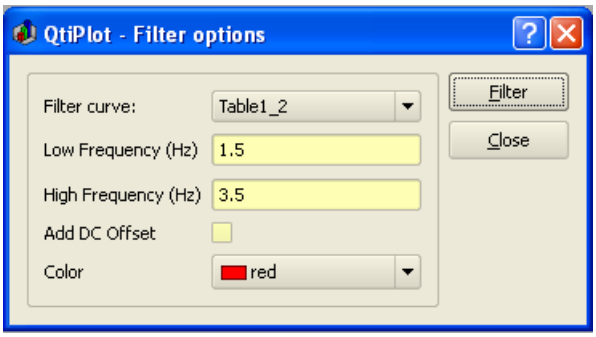


Figure 3.6: The **FFT Filter -> Band Pass...** dialog.

This command creates a new table with the filtered data, and a new curve will be added on the current plot.

Band Block... 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.

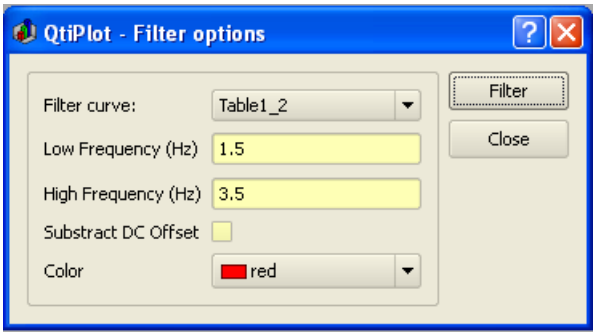


Figure 3.7: The **FFT Filter -> Band Block...** dialog.

This command creates a new table with the filtered data, and a new curve will be added on the current plot.

Analysis -> 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 Akime spline* (the curve you analyse must contain at least 5 points). See the Analysis chapter for a comparison of the differents methods.

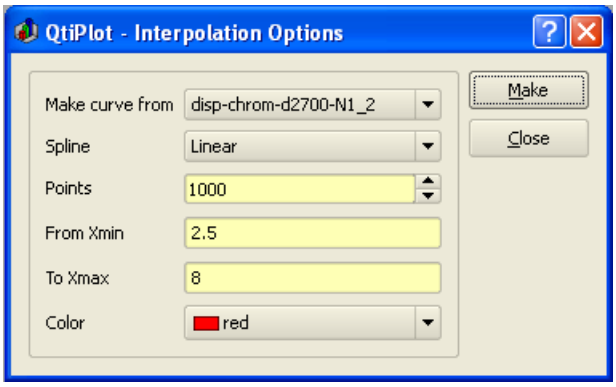


Figure 3.8: The **Interpolate...** dialog.

This command creates a new curve on the current plot, and a new table.

Analysis -> FFT... Performs a forward or inverse FFT transform of the selected curve. The parameters used can be set with the FFT dialog.

The inverse FFT transform of a forward transform will result in a data set identical to that used for the forward transform.

Analysis -> Fit Linear Performs a linear fit of the selected curve. The results will be given in the Log panel

Analysis -> 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.

Analysis -> 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.

Second Order... Opens a dialog, allowing you to choose the curve to fit and the initial guesses for the fit parameters.

Third Order... Opens a dialog, allowing you to choose the curve to fit and the initial guesses for the fit parameters.

Analysis -> Fit Exponential Growth... Performs an exponential growth fit of the selected curve.

Analysis -> 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.

Analysis -> 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.

Analysis -> Fit Bolzmann (sigmoidal) 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.

Analysis -> Fit Multi-peak ->Gaussian... Performs a fit to a sum of N gaussian functions of the selected curve.

Analysis -> Fit Multi-peak -> Lorentzian... Performs a fit to a sum of N lorentz functions of the selected curve.

3.9 The Table Menu

This menu is only active when a table is selected.

Set Column As These commands are used to define the kind of data which is stored in the different columns of 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.
- **Set Column As -> Y error** Define the selected column for use as error bars for Y-values.
- **Set Column As -> Read-only** Set the selected columns as read-only.
- **Set Column As -> Read/Write** Restore the write acess to the selected columns.
- **Set Column As -> Disregard** The selected column can be used in different ways in several plots (as X values, Y values, etc).

Column Options... This command is used to define the global parameters of each column such as numeric format, column name, etc. See the corresponding dialog box section for more details.

Set Column Values... This command is used to fill the selected column with the values resulting from a mathematical formula. See the corresponding dialog box section for more details.

Recalculate When you fill a column (named for example 'C1') with the results of a formula (by using the Set Column Values... 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.

Fill column with This command is used to fill the selected column with special values:

Fill Column With -> Row Numbers The filling is done with the number of the corresponding rows.

Fill Column With -> Random Numbers The filling is done with random values between 0 and 1.

Clear Removes all 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 left column.

Columns 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.

Move to First Moves the selected column to the beginning of the table.

Move Left Moves the selected column to the left.

Move Right Moves the selected column to the right.

Move to Last Moves the selected column to the end of the table.

Swap columns Swap the selected columns.

Rows 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 Row... Defines the active line in the selected table.

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.

3.10 The Matrix Menu

This menu is only active when a matrix is selected.

Set Properties... This command opens a dialog window which is used to specify some view parameters of the matrix (cell width, format of numbers).

Set Dimensions... 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.

Set Values... This command opens a dialog window which is used to fill in a matrix with the result of a function z=f(i,j) in which i and j stand for the row and column numbers.

Recalculate (Ctrl-Return) This command allows to recalculate the matrix cell values using a predefined formula.

Rotate 90 (Ctrl-Shift-R) This command performes a clockwise 90 degrees rotation of the active matrix.

Rotate -90 (Ctrl-Alt-R) This command performes a counterclockwise 90 degrees rotation of the active matrix.

Flip V (Ctrl-Shift-V) Flips vertically the selected matrix.

Flip H (Ctrl-Shift-H) Flips horizontally the selected matrix.

Transpose Transpose the selected matrix.

Invert Inverse the selected matrix.

Determinant Compute the determinant of the selected matrix.

Go to Row... (Ctrl-Alt-G) This command opens a dialog allowing to select a row index that will become the current row in the selected matrix.

View

Image mode (Ctrl-Shift-I) Displays the selected matrix as an image.

Data mode (Ctrl-Shift-D) Displays the selected matrix as a data table.

Palette Gray Scale Map If the selected matrix is viewed as an image, this command sets its palette to a gray scale.

Rainbow If the selected matrix is viewed as an image, this command sets its palette to a predefined color scale.

Custom Opens a dialog allowing to customize the palette of the selected matrix, in case this is displayed as an image.

Show Column/Row (Ctrl-Shift-C) By checking this option the horizontal and vertical headers of the selected matrix will display the column/row indexes.

Show X/Y (Ctrl-Shift-X) By checking this option the horizontal and vertical headers of the selected matrix will display the x/y coordinates.

Convert to Spreadsheet Convert the selected matrix in a table.

3.11 The Format Menu

This menu is only active when a plot is selected.

Plot... In the case of a classical 2D plot, opens the format plot dialog with the general plot options tab selected. It allows to customize the line styles and colors of plot frame, etc.

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.

Curves... Opens the Custom Curves dialog. It allows to customize the line style and colors used to draw curves.

If the selected plot is a surface plot, this menu item is not showed.

Scales... 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.

Axes... 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.

In the case of a surface plot, this command opens the surface plot options with the axis options tab selected.

Grid... Opens the format plot dialog with the grid tab selected. It allows to add and customize grid lines on the different axes.

If the selected plot is a surface plot, this menu item is not showed.

Title... Opens a text options dialog, allowing you to modify the title of the plot and its properties (color, font, alignement).

In the case of a surface plot, this command opens the surface plot options with the title options tab selected.

3.12 The Scripting Menu

Scripting -> Scripting language Opens a dialog allowing to choose the scripting language for the current project.

Scripting -> Restart Scripting Reinitializes the scripting environement.

Scripting -> Add Custom Script Action... Opens the Custom Action dialog, which allows you to define menu items and toolbar buttons used to launch Python scripts.

If the actif window in the project is a Notes window, the following items are also available in the menu:

Scripting -> Execute (Ctrl+J) Executes the line where the mouse cursor is placed in the Notes window.

Scripting -> Preferences... (Ctrl+Shift+J) Executes all lines in the Notes window.

Scripting -> Evaluate (Ctrl+Return) Evaluates the line where the mouse cursor is placed in the Notes window.

3.13 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.

Folders Opens a menu displaying a list of all the folders and subfolders in the project. The active folder is toggled. You can change the active folder in the project by selecting an item from this list.

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.

Script Window (F3) Opens the script console 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).

3.14 Customization of 3D plots

Theses commands are not available through any menu nor by any keyboard shortcut. They can be accessed through the 3D toolbar.

Frame This command can be accessed by a click on the 

Draws only the three axis on the active 3D plot.

Box This command can be accessed by a click on the 

Draws the three axis on the active 3D plot, and a box around it.

No axes This command can be accessed by a click on the 

Doesn't draw the three axis nor the box on the active 3D plot.

Front Grid This command can be accessed by a click on the 

Draws a grid on the front panel of the active 3D plot. The position of this grid is the plan defined by $y=y_{min}$.

Back Grid This command can be accessed by a click on the 

Draws a grid on the back panel of the active 3D plot. The position of this grid is the plan defined by $y=y_{max}$.

Left Grid This command can be accessed by a click on the 

Draws a grid on the left panel of the active 3D plot. The position of this grid is the plan defined by $x=x_{min}$.

Right Grid This command can be accessed by a click on the 

Draws a grid on the right panel of the active 3D plot. The position of this grid is the plan defined by $x=x_{max}$.

Ceiling Grid This command can be accessed by a click on the 

Draws a grid on the top panel of the active 3D plot. The position of this grid is the plan defined by $z=z_{max}$.

Floor Grid This command can be accessed by a click on the 

Draws a grid on the floor panel of the active 3D plot. The position of this grid is the plan defined by $z=z_{min}$.

Enable perspective This command can be accessed by a click on the 

Enables/Disables the 3D perspective mode.

Reset rotation This command can be accessed by a click on the 

Resets the rotation of the 3D plot to the default values.

Fit frame to window This command can be accessed by a click on the 

Finds the best layout of the 3D plot fitting the window size. It readjusts the length of the axis ticks to a default value.

Bars Style This command can be accessed by a click on the 

If the active 3D plot is a 3D histogram, this command is used to modify the style of the bars.

Dots This command can be accessed by a click on the 

If the active 3D plot is a 3D scatter, this command is used to modify the style of the data points to dots.

Cones This command can be accessed by a click on the 

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 This command can be accessed by a click on the 

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 This command can be accessed by a click on the 

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 This command can be accessed by a click on the 

If the active 3D plot is a 3D surface, this command is used to modify the style of the surface to a wireframe. A computation of the hidden line is done.

3D Polygons This command can be accessed by a click on the 

If the active 3D plot is a 3D surface, this command is used to modify the style of the surface to polygons.

3D Wire Surface This command can be accessed by a click on the 

If the active 3D plot is a 3D surface, this command is used to modify the style of the surface to polygons with a mesh.

Floor Data Projection This command can be accessed by a click on the 

If the active 3D plot is a 3D surface, this command is used to add a filled area projection of the surface on the floor of the plot.

Floor Isolines This command can be accessed by a click on the

If the active 3D plot is a 3D surface, this command is used to add an isoline.

Empty Floor This command can be accessed by a click on the 

If the active 3D plot is a 3D surface, this command is used to remove any projection from the floor.

Animation This command can be accessed by a click on the 

Enables/disables animation.

Chapter 4

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/hidding 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.

4.1 The Edit Toolbar

Figure 4.1: The QtiPlot Edit Toolbar

4.2 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.

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		Delete the current selection.

Table 4.1: Edit toolbar commands.

Figure 4.2: The QtiPlot File Toolbar

4.3 The Plot Toolbar.

This toolbar is only active when a plot window is selected. It allows the quick access to the commands of the Graph menu and of the Data menu which are used for the modification of the plots and of the data points of the plots.

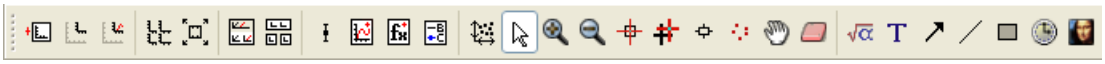


Figure 4.3: The QtiPlot Plot Toolbar

4.4 The Table Toolbar

This toolbar allows a quick access to the commands of the Plot Menu used to plot the datas of the table.

Icon	Command	Key	Description
	New -> New Project command	Ctrl-N	Create a new project.
	New -> New Folder command	F7	Add a new folder.
	New -> New Table command	Ctrl-T	Create a new table.
	New -> New Matrix command		Create a new matrix.
	New -> New Note command		Create a new note window.
	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 Surface 3D 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 QtiPlot project file.
	Append Project... command	Ctrl-Alt-A	Appends an existing QtiPlot project file as a new folder to the currend project.
	Open Template command		Opens an existing template QtiPlot project file.
	Save Project command	Ctrl-S	Saves the current project.
	Save as Template command		Saves the current project as a template.
	Import -> Import ASCII... command		Import ASCII files.
	Duplicate command		Clonates the active window.
	Print command	Ctrl-P	Print the active window.
	Export to PDF command		Export the active window to the PDF file format.
	Project Explorer command	Ctrl-E	Show or hide the project explorer.
	Results log command		Show or hide the results window.
	Script Window command	F3	Show the script window.

Table 4.2: File toolbar commands.

Icon	Command	Key	Description
	Add Layer	Alt-L	Adds a new layer to the active plot window.
	Add Inset Layer		Adds an empty inset layer to the active plot window.
	Add Inset Layer		Adds an inset layer containing the curves in the active plot layer.
	Arrange Layers	Shift-A	Arranges the different layers of the active plot window.
	Automatic Layout		Automatically arranges the different layers of the active plot window.
	Extract to Layers		Extracts datasets in the active layer to separate layers.
	Extract to Graphs		Extracts all layers in the active graph window to separate graphs.
	Add/Remove Curves...	Alt-C	Adds or removes curves to the active plot window.
	Add Error Bars...	Ctrl-B	Adds error bars to a curve of the active plot window.
	Add Function...	Ctrl-Alt-F	Adds a curve based on a function to the active plot window.
	New Legend	Ctrl-L	Adds a new legend to the active plot window.
	Rescale to Show All	Ctrl-Shift-R	Rescales the axes to show all data points.
	Disable tools		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.
	Zoom in	Ctrl-+	Switches the active plot layer to the zoom mode.
	Zoom out	Ctrl--	Switches the active plot layer to the zoom mode.
	Data Reader	Ctrl-D	Switches the data display mode.
	Select Data Range	Alt-S	Switches the active plot to the **Select Data Range** mode.
	Screen Reader		Switches the active plot layer to the **Screen Reader** mode.
	Move Data points	Ctrl-Alt-M	Allows to move data points on the active plot.
	Remove Bad Data Points	Alt-B	Allows to remove data points on the active plot.
	Add Equation command	Alt-Q	Adds a new Tex equation to the active plot.
	Add Text command	Alt-T	Adds a new text element in the active plot.
	Draw Line command	Ctrl-Alt-L	Adds a new line on the active plot.
	Draw Arrow command	Ctrl-Alt-A	Adds a new arrow on the active plot.
	Add Rectangle command	Ctrl-Alt-R	Adds a new rectangle to the active plot.
	Add Time Stamp command	Ctrl-Alt-T	Adds a time/date label on the active plot.
	Add Image command	Alt-I	Inserts a new image in the active plot.

Figure 4.4: The QtiPlot Table Toolbar

Icon	Command	Key	Description
	plot -> Line		plot with the line style.
	plot -> Scatter		plot with the scatter style.
	plot -> Line+Symbol		plot with the line+symbol style.
	plot -> Columns		plot with the columns style.
	plot -> Rows		plot with the rows style.
	plot -> Area		plot with the area style.
	plot -> Pie		plot with the pie style.
	plot -> Statistical Graphs -> Histogram		plot with the histogram style.
	plot -> Vectors XYXY		plot with the vector style.
	plot -> Double-Y		plot with two different y axes.
	plot -> Zoom		plots a magnified region of data in the same graph window (along with the full range of data).
	plot -> Plot 3D -> Ribbons		plot with the 3D ribbons style.
	plot -> Bars		plot with the 3D bars style.
	plot -> Scatter		plot with the 3D scatter style.
	plot -> Plot 3D -> Trajectory		plot with the trajectory style.
	plot -> Add Column	Alt + C	add a new column to the table
	plot -> Statistics on Columns		compute statistical parameters on selected columns
	plot -> Statistics on Rows		compute statistical parameters on selected row

Table 4.4: Table toolbar commands.

4.5 The Column Toolbar

This toolbar allows a quick access to the commands of the Table Menu used to rearrange columns in the table or to define the plot role of the selected columns.

Figure 4.5: The QtiPlot Column Toolbar

Icon	Command	Description
X	table -> Set Column As -> X	Define the selected column as abscissae for the plots.
Y	table -> Set Column As -> Y	This command defines the selected column as ordinates for the plots.
Z	table -> Set Column As -> Z	In the case of 3D plots, Z columns will be used as height values.
	table -> Set Column As -> Y error	Define the selected column for use as error bars for Y-values.
NONE	table -> Set Column As -> Disregard	This command removes any plot role from the selected columns.
	table -> Move to First	Moves the selected column to the beginning of the table.
	table -> Move Left	Moves the selected column to the left.
	table -> Move Right	Moves the selected column to the right.
	table -> Move to Last	Moves the selected column to the end of the table.
	table -> Swap columns	Swap the selected columns.

Table 4.5: Column toolbar commands.

4.6 The Plot 3D Toolbar

Figure 4.6: The QtiPlot Plot 3D Toolbar

Icon	Command	Key	Description
	plot -> Frame		Draw only the three axes.
	plot -> Box		Draw the three axes and the 3D box around the plot.
	plot -> No axes		Doesn't draw the axes nor the box.
	plot -> Front Grid		Draw a grid on the front panel.
	plot -> Back Grid		Draw a grid on the back panel.
	plot -> Left Grid		Draw a grid on the left panel.
	plot -> Right Grid		Draw a grid on the right panel.
	plot -> Ceiling Grid		Draw a grid on the top panel.
	plot -> Floor Grid		Draw a grid on the bottom panel.
	plot -> Floor Grid		Draw a grid on the bottom panel.
	plot -> Enable perspective		Enables/Disables the 3D perspective mode.
	plot -> Reset rotation		Resets the rotation of the 3D plot to the default values.
	plot -> Fit frame to window		Finds the best layout of the 3D plot fitting the window size. It readjusts the length of the axis ticks to a default value.
	plot -> Bars Style		Changes the styles of the bars.
	plot -> Dots		Draw the 3D scatter points with the dot style.
	plot -> Cones		Draw the 3D scatter points with the cone style.
	plot -> Cross Hairs		Draw the 3D scatter points with the cross-hairs style.
	plot -> 3D Wire Frame		Draw a surface with the wireframe style.
	plot -> 3D Hidden Lines		Draw a surface with the mesh style (with hidden lines).
	plot -> 3D Polygons		Draw a surface with the polygons style.
	plot -> 3D Wire Surface		Draw a surface with the mesh+polygons style.
	plot -> Floor Data Projection		Draw a projection of the plot on the floor.
	plot -> Floor Isolines		Draw an isolines projection on the floor.
	plot -> Empty Floor		Draw an empty floor.
	plot -> Animation		Enables/Disables animation.

Table 4.6: 3D Plot toolbar commands.

Chapter 5

The Dialogs

5.1 Add Custom Action

This dialog helps you define new Python script launchers that will be added to menus or tool bars in QtiPlot. You can fully customize the script launchers by defining: a textual description that will be displayed in the destination menu, an icon, a shortcut key sequence and a tool tip. The *Tool Tip Text* is a short description that will be displayed near the launcher button when you hover the mouse over it. You can only assign unique shortcut key sequences to your custom launchers.

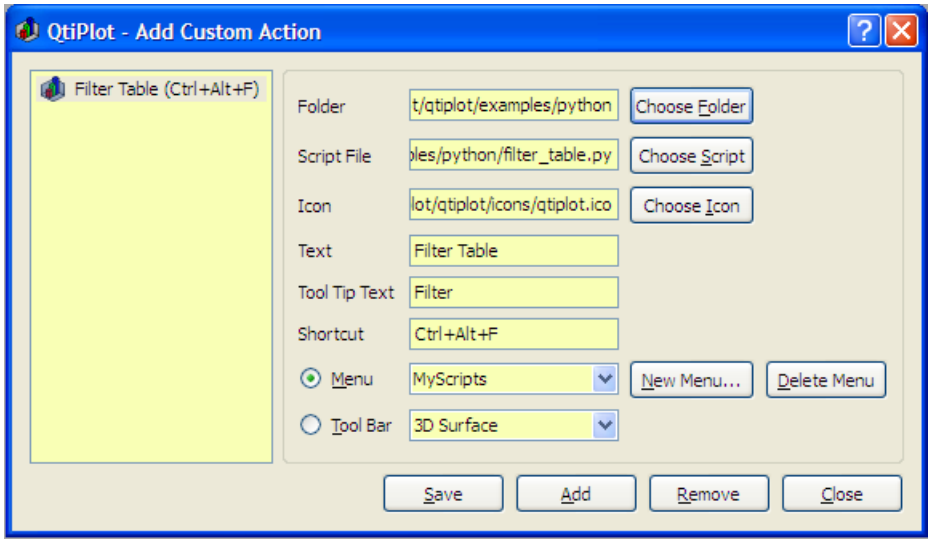


Figure 5.1: The **Add Custom Script Action...** dialog box.

The *Script File* is the path to the Python script that will be executed when you click

the launcher. When you click the *Add* button all the settings for the new launcher are saved to an XML file in the directory specified in the *Folder* box. On start-up QtiPlot parses all XML files in this folder and creates the corresponding menu items or toolbar buttons. When you press the *Remove* button, the launcher selected in the left side list of the dialog is removed from QtiPlot menus/tool bars and the corresponding XML file is deleted from the hard disk.

5.2 Add Error bars

This dialog is activated by selecting the Add Error Bars... command from the Graph menu.

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.

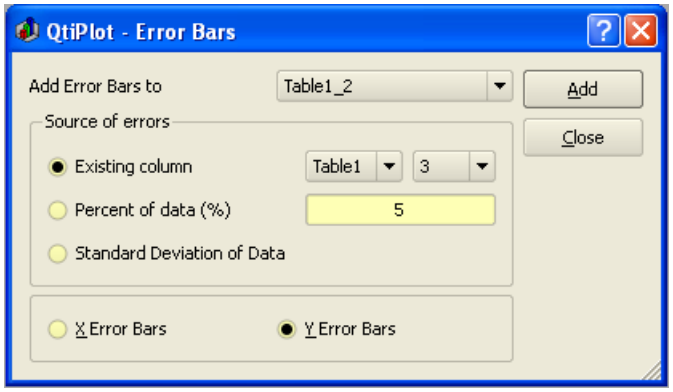


Figure 5.2: The **Add Error Bars...** dialog.

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.

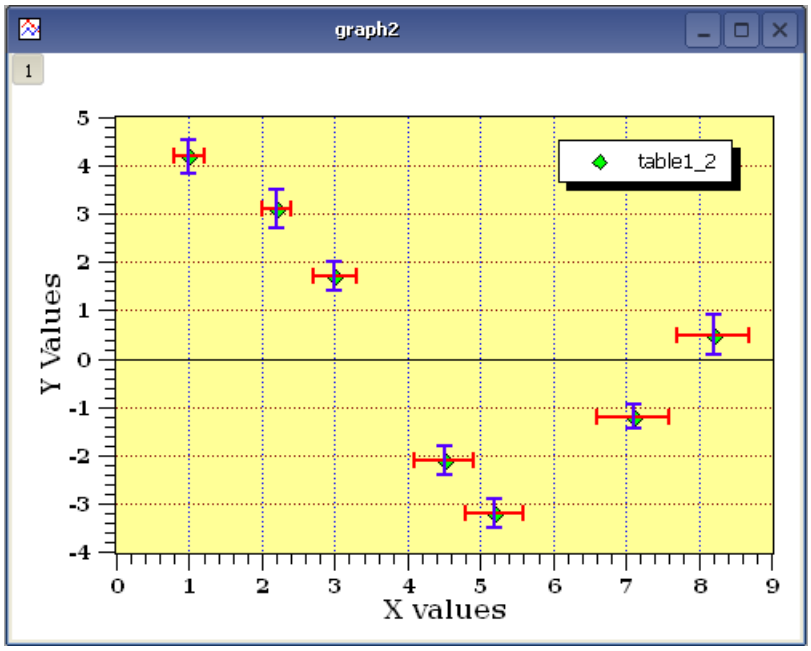


Figure 5.3: A plot with X and Y Error Bars.

5.3 Add Function

This dialog box is used to add a function curve to the active plot. The function can be built with the common operators: * + / - and ^ 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.

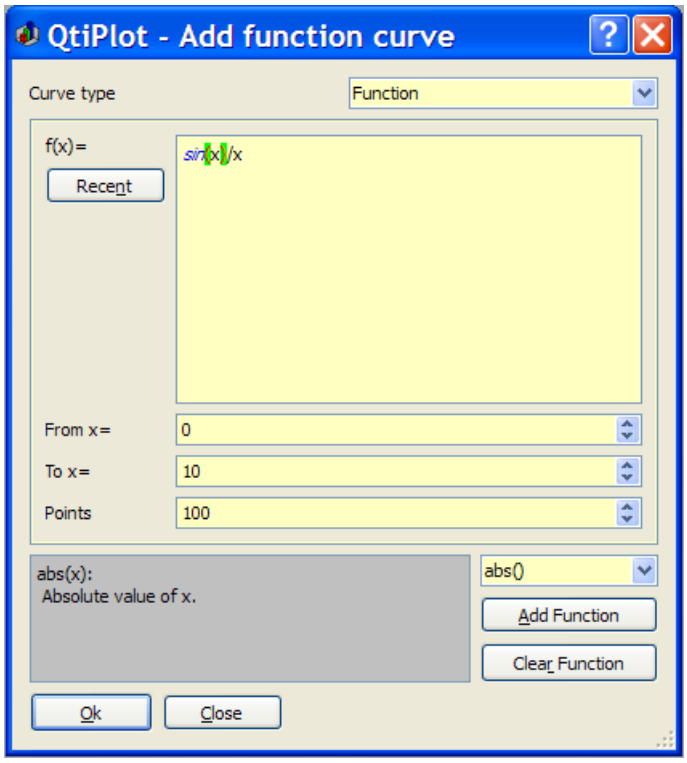


Figure 5.4: The **Add Function...** dialog box: cartesian coordinates.

If the function expression contains constants, e.g.: f(x) = a*x + b, QtiPlot automatically detects them and displays a two column table on the right side of the dialog, containing input spin boxes that allow to choose appropriate values for those constants:

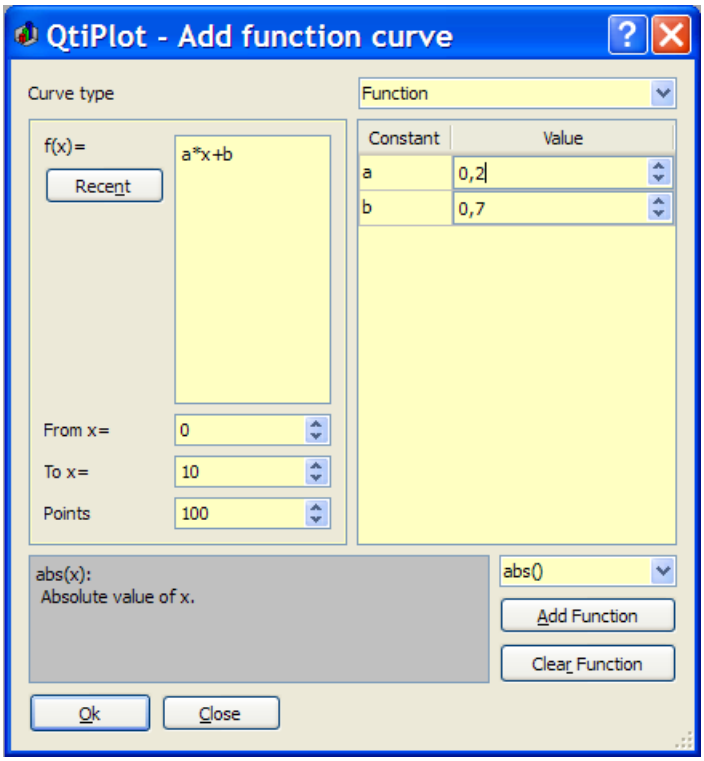


Figure 5.5: The **Add Function...** dialog box: automatic detection of constants.

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.

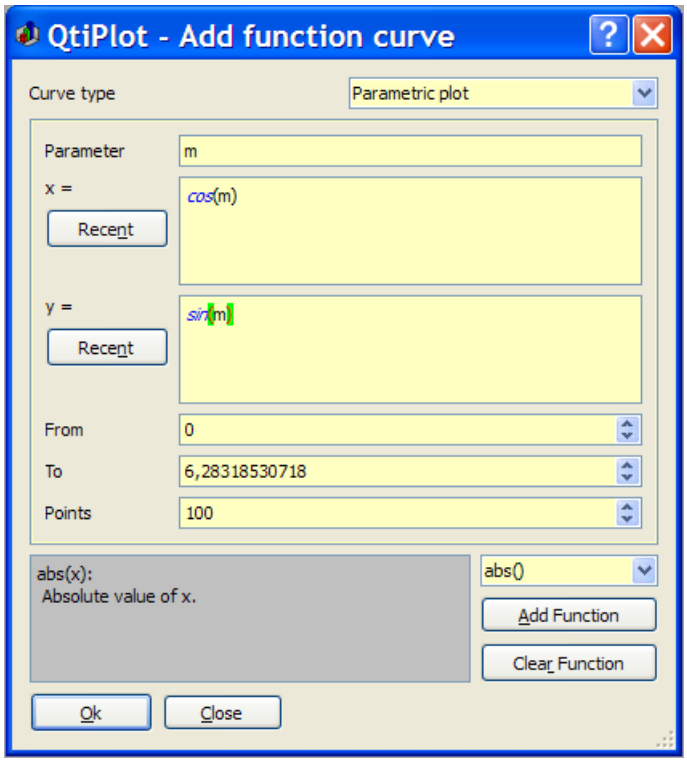


Figure 5.6: The **Add Function...** dialog box: parametric coordinates.

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.

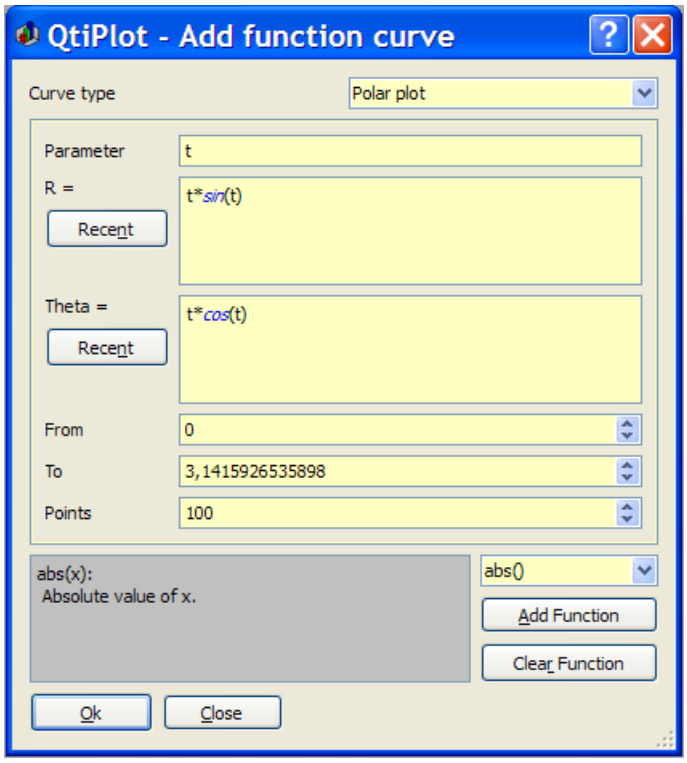


Figure 5.7: The **Add Function...** dialog box: polar coordinates.

5.4 Add Layer

This dialog is opened when you want to add a new layer on the active plot. If you select *Guess*, QtiPlot will divide the window in two columns and put the new layer on the right. If you choose *Top-Left Corner*, QtiPlot will create a new layer with the maximum possible size over the existing layer, this layer contains an empty plot. 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.

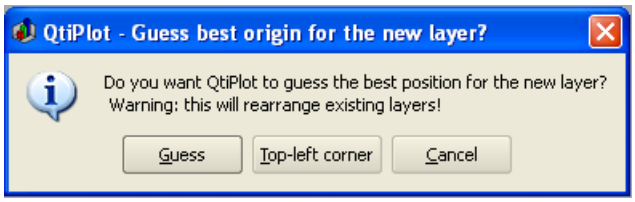


Figure 5.8: The **Add Layer** dialog box.

5.5 Add/Remove curves

This dialog is activated by selecting the command Add/Remove Curves... from the Graph Menu.

The left window shows the columns which are available for plotting in the different tables of the project. If the *Show current folder only* option is checked, only the tables in the active folder of the project will be listed in this window. The right window gives the list of the curves already plotted. In the case presented below, there are two tables from which the **Add/Remove Curves...** dialog box allows to select columns. If you use this dialog box to add a column, the X column will be the one defined as X in the corresponding table.

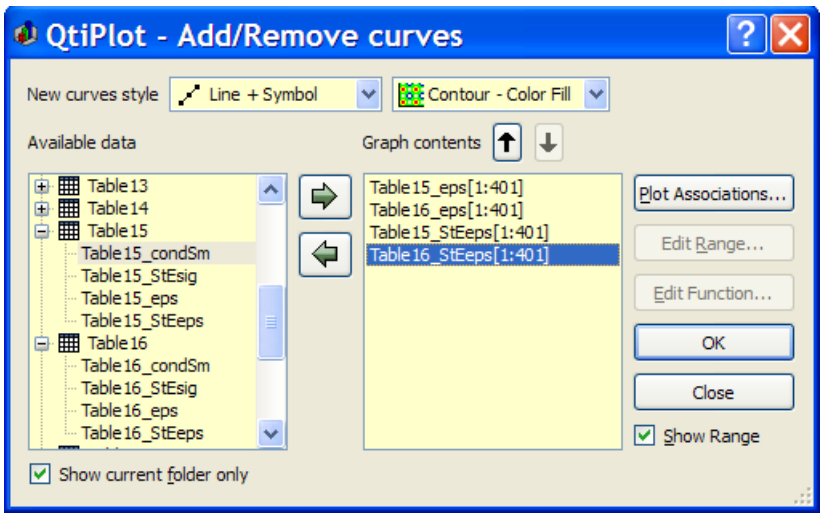


Figure 5.9: The **Add/Remove Curves...** dialog box.

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 want

to do this, you have to copy the columns in the same table.

If the curve selected is a function, you can modify it by pressing the *Edit Function...* button. Refer to the Add Function... dialog box for more details on functions editing.

If the *Show Range* option is checked, the plot range of all the curves not defined as analytical functions will be also displayed in the right window of the dialog. The plot range is the interval of points which are visible in a curve. You can show/hide data points from a selected curve, without having to delete them, by pressing the *Edit Range...* button, which opens a dialog allowing to edit the plot range.

5.6 Arrange Layers

This dialog is activated by selecting the command **Arrange Layers** from the Graph Menu or by the key code Shift-A.

It allows to modify the geometrical arrangement of the plots which are already present in the active window. You can add new layers or remove existing ones using the *Number* of layers control. By checking the *Link X axes* option, the layers will be interconnected, meaning that if you manually set the abscissae scale limits for one plot layer, the same limits will be automatically set for all other layers in the plot window.

You can specify the numbers of rows and columns which will define a grid of plots. With the default settings, QtiPlot 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 drawing area of the layers and QtiPlot will modify the size of the window. By checking the *Fixed size* box, the size of the layers will be kept unchanged when you manually resize the plot window.

The controls in the two right group boxes, *Alignment* and *Spacing*, allow to set the alignment of the layers in the window, and the margins between the layer borders and the window limits. By choosing, the option *Canvases* for the *Spacing* | *Align* field, the spacing defines the distance between the axes of the layers, whereas with the *Layers* option it defines the distance between the borders of the layers.

Finally you have the possibility to swap two layers using the *Swap Layers* box.

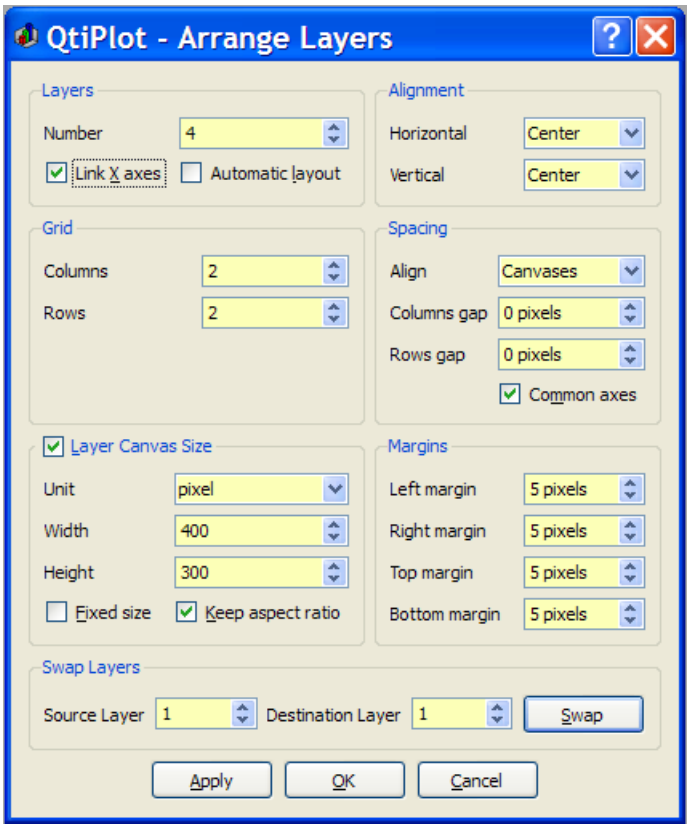


Figure 5.10: The **Arrange Layers** dialog: the geometry tab

The layers will be arranged in order to obtain a good alignment of the vertical and horizontal axis.

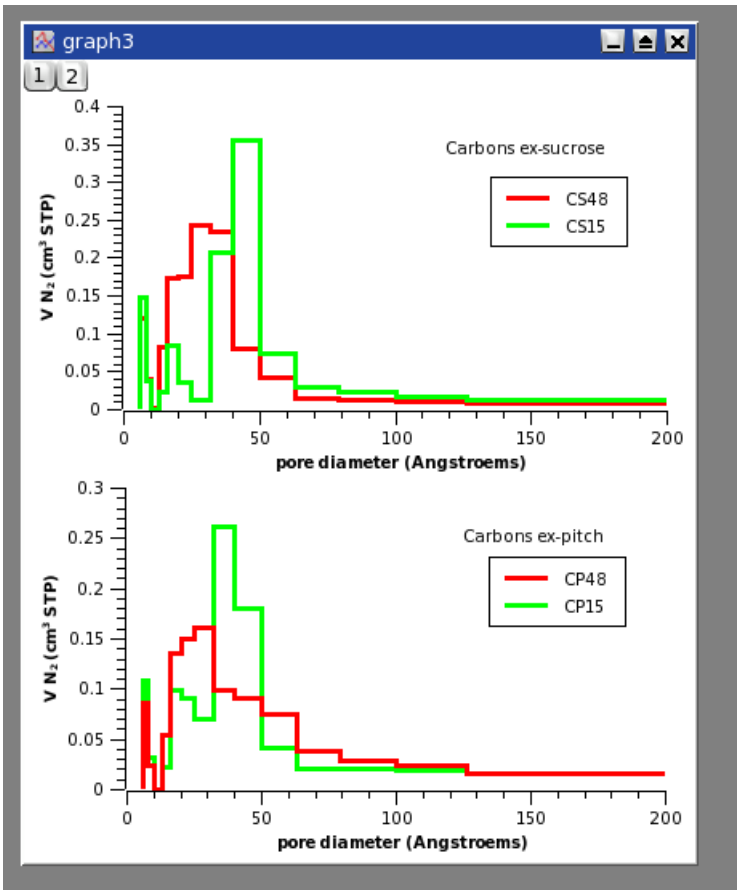


Figure 5.11: Exemple of a vertical arrangement for two plots.

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.

5.7 Line Options

This dialog allows to modify a line or an arrow which has been created by the command ***Draw Arrow*** from the Graph Menu or with Ctrl-Alt-A. One can also 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.

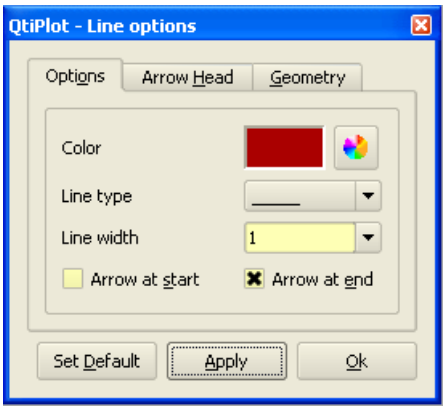


Figure 5.12: The *Arrow options* dialog: first tab

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.

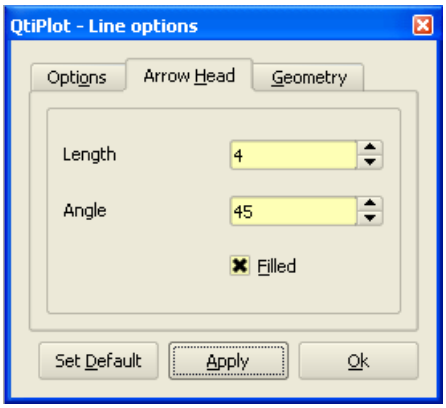


Figure 5.13: The *Arrow options* dialog: second tab

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.

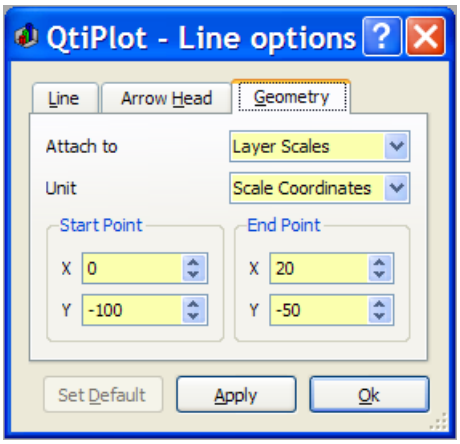


Figure 5.14: The *Geometry* dialog: third tab

5.8 Column Options

This dialog is activated by selecting the command Column Options... from the Table Menu. At least one column must be selected.

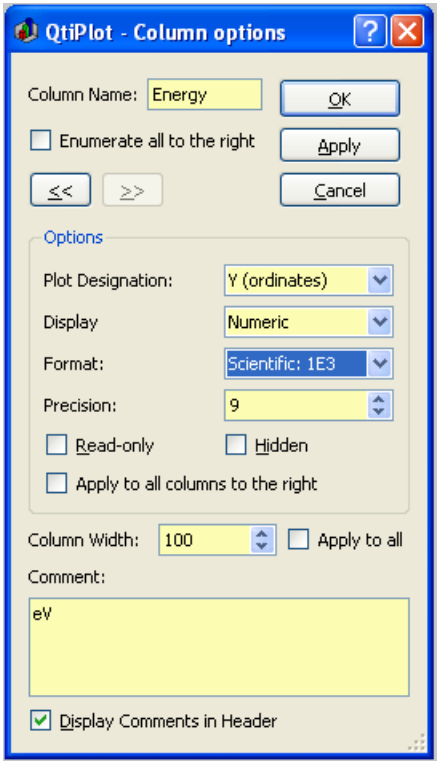


Figure 5.15: The **Column Options...** dialog.

The checkbox *Enumerate all to the right* can be used to build the name of all the columns which are at the right of the selected one. If the name of the selected column is "xyz", this column and the following ones will be renamed to "xyz1", "xyz2", and so on.

The buttons "<<" and ">>" are used to change the selected column. The highlighting of the column in the table behind the dialog box will change indicating that a new column was selected. The column to which the formatting commands are applied is the one whose name appear in the "Column Name" box.

The *Plot Designation* selector is used to define the columns which are used as X, Y or Z values or as error bars. In a table you can select several columns as X, in this case in the column label they will be indicated as [X1], [X2], etc... and their corresponding Y columns will be indicated as [Y1], [Y2], etc...

The *Comment* box allows to enter a text that can be displayed in the table header under the column name, if the option *Display Comments in Header* is checked.

5.9 Contour Curves Options

This dialog is activated by clicking on a contour curve (or on the plotting area) when a 3D plot has been created from a matrix with one of the following commands of the Plot 3D menu: Contour+Color Fill command, Countour Lines command or Gray Scale Map command.

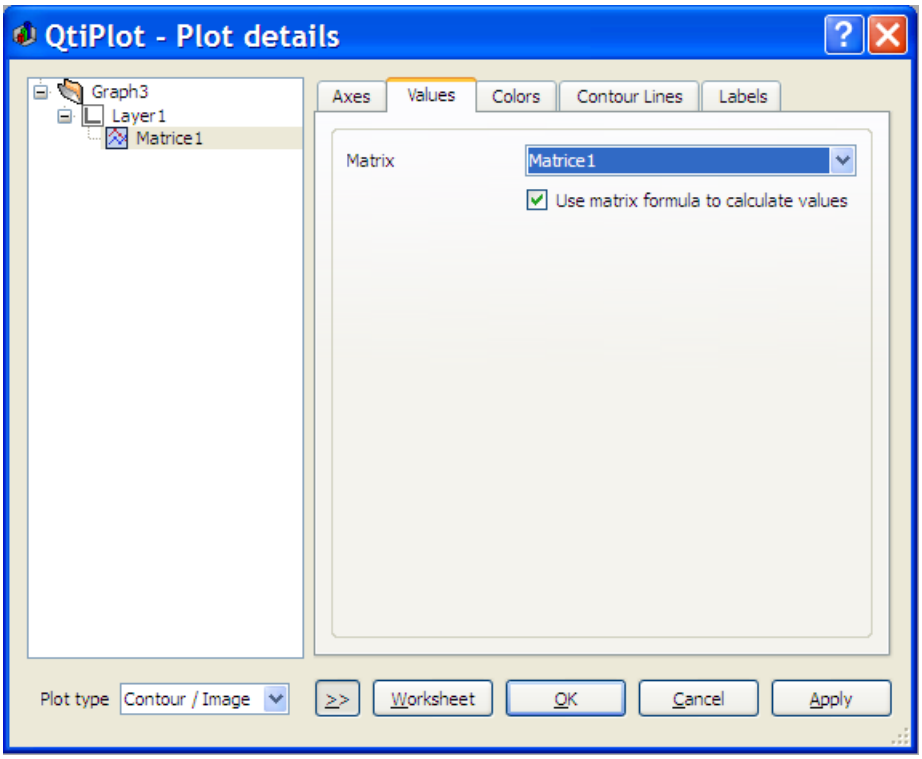


Figure 5.16: The Values tab.

The *Values* tab allows you to choose the matrix which is the data source of the plot. It is possible to use the formula defined for the matrix in order to calculate the Z values to be displayed, by checking the box *Use matrix formula to calculate values*. This results in a higher accuracy, especially when drawing the contour lines, but it only works if the formula uses a muParser compatible syntax.

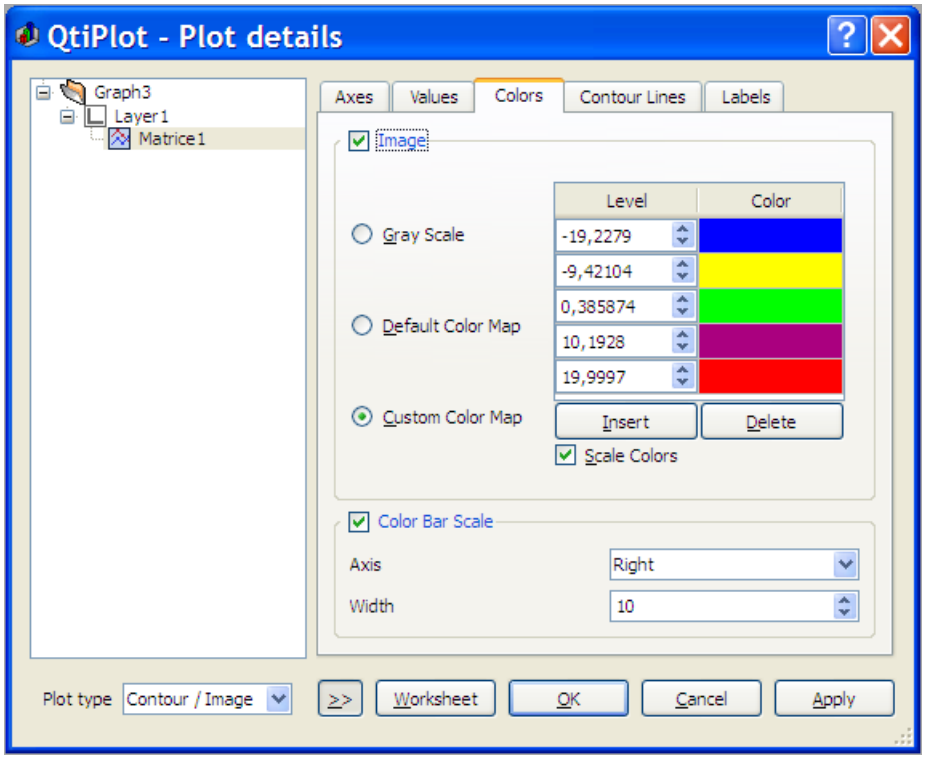
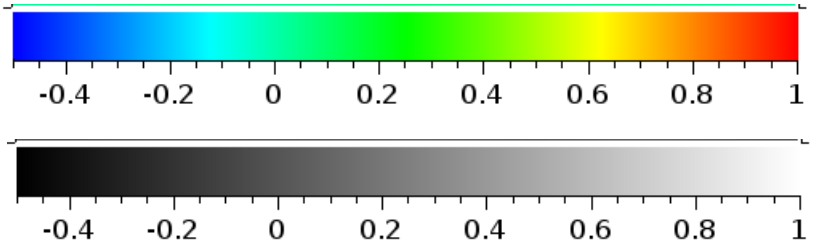
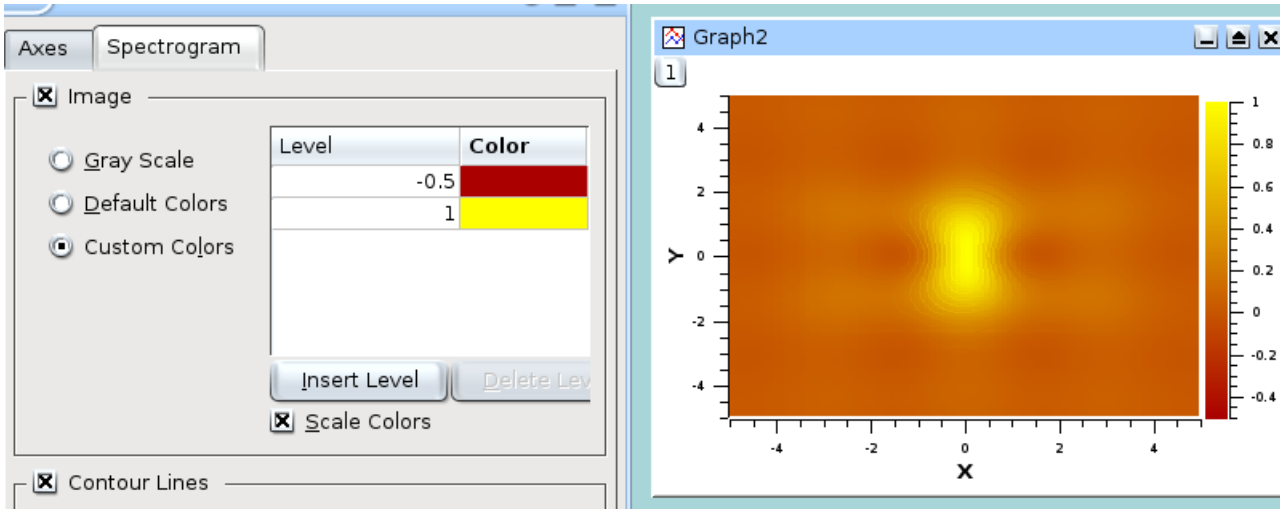


Figure 5.17: The Colors tab.

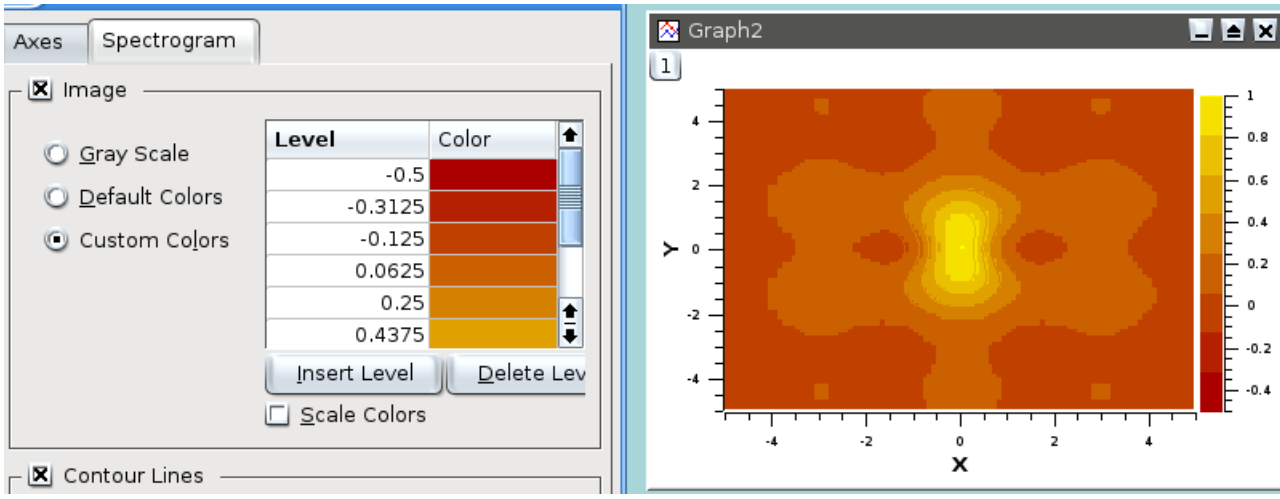
The first group of settings *Image* is checked if you want to have a color or gray level filling of the contour plot. The default gray and color maps are the following:



You can customize this colormap by checking the *Custom Colors* box. A table with a set of numbers (the Z levels) and the corresponding colors is presented. You can then add or delete new levels for the definition of the colormap, and modify the corresponding Z levels. You are not allowed to modify the first and last levels, which are set to the minimum and maximum Z values. Beware that this is only the definition of the colormap, it won't change the number of contour lines of your plot. An example of classical custom colormap is given here:



If you want to obtain discrete colors for each level,you must uncheck the *Scale Colors* checkbox. In this case you must define enough levels in your colormap.



The second group of settings must be checked if you want to have a bar scale on your plot. You can then define its position and width.

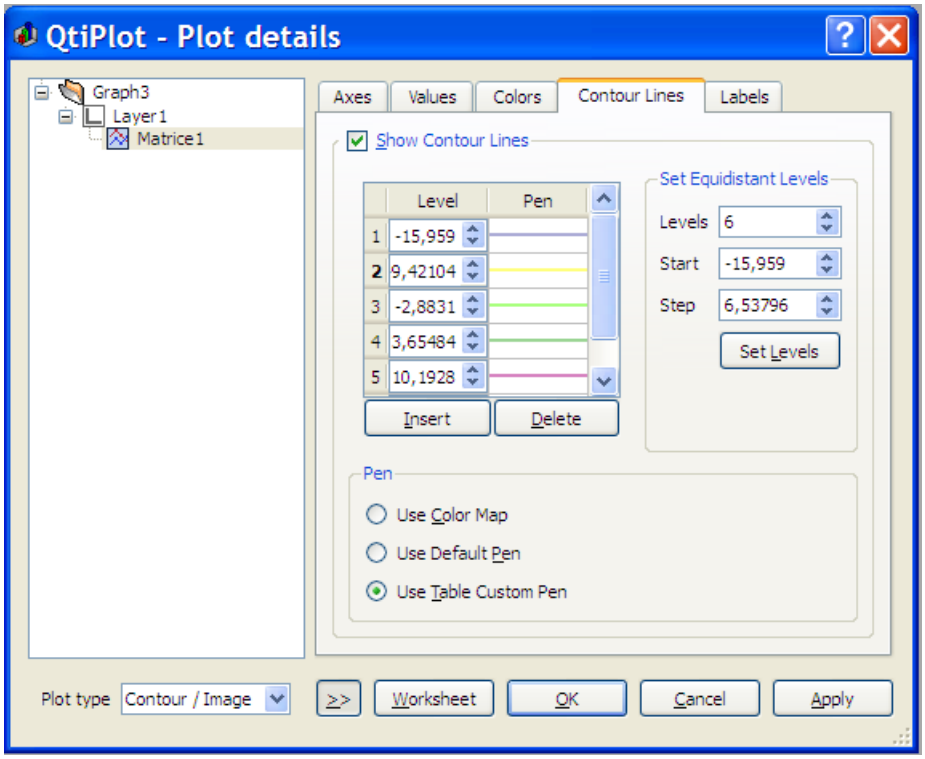


Figure 5.18: The Contour Lines tab.

The *Contour Lines* tab is used to customize the settings of contour lines. You can select the number of lines and their color. If you check *Use Default Pen*, the color of the line will follow the settings defined in the group at the left of the checkbox. If you check *Use Color Map*, the lines will be colored as a function of the Z levels following the colormap defined in the *Colors* setting tab. If you check *Use Table Custom Pen*, the color of the line will follow the settings defined in the *Pen* column of the table displaying the Z levels.

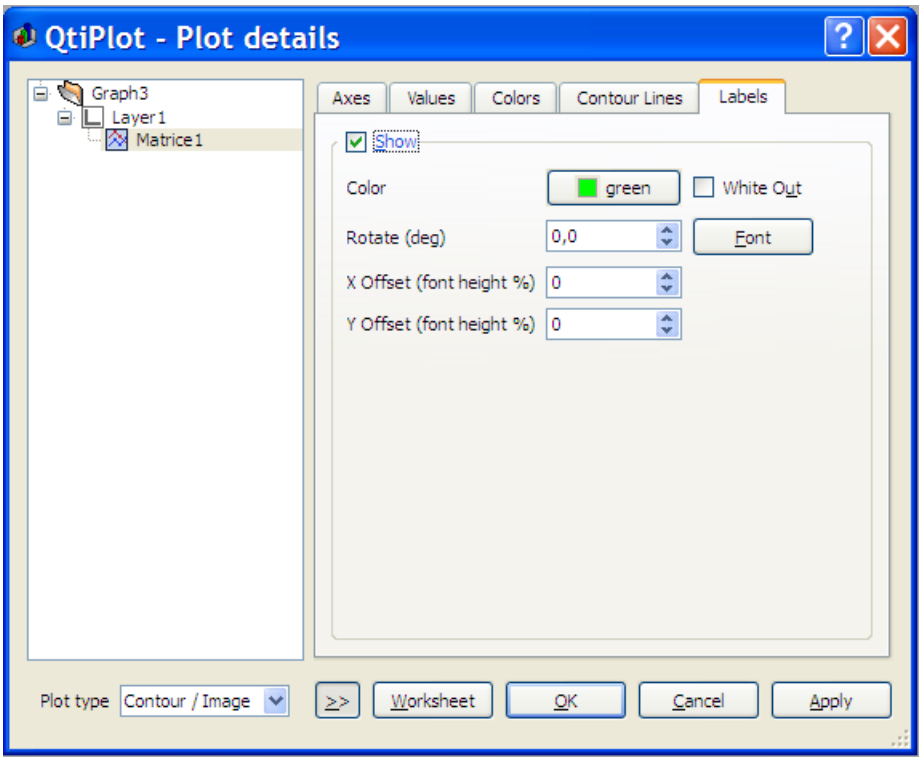


Figure 5.19: The Labels tab.

The *Labels* tab is used to enable/disable the display of the Z value for each of the contour lines. You can globally customize the color, background, font, rotation and position of the text labels.

5.10 Plot Details

This dialog is activated by selecting the command Plot... from the Format Menu. It is also activated by a double click on the plot. If there are more than one layer in the window, QtiPlot will select the layer which contain the plot under the mouse pointer.

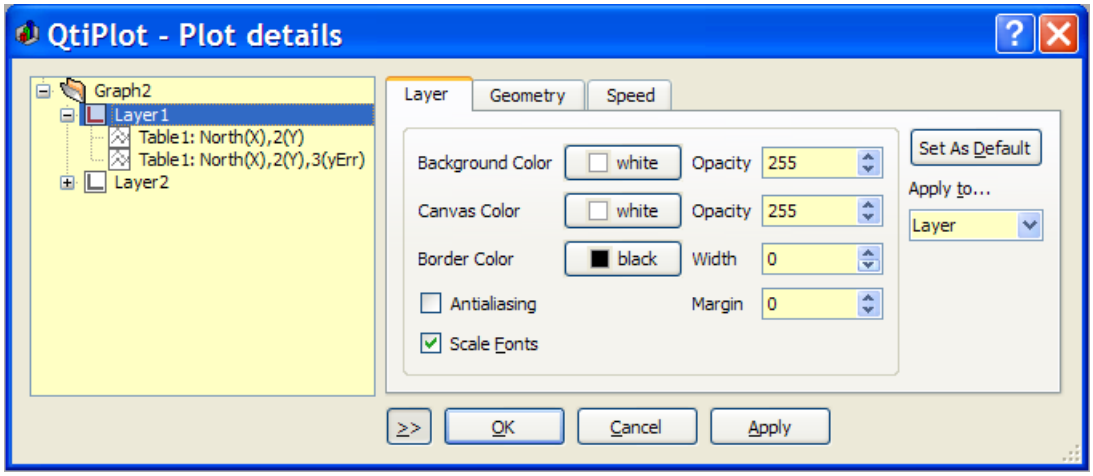


Figure 5.20: The Plot Details Dialog: Layer properties.

The second tab defines the geometry of the plot layer.

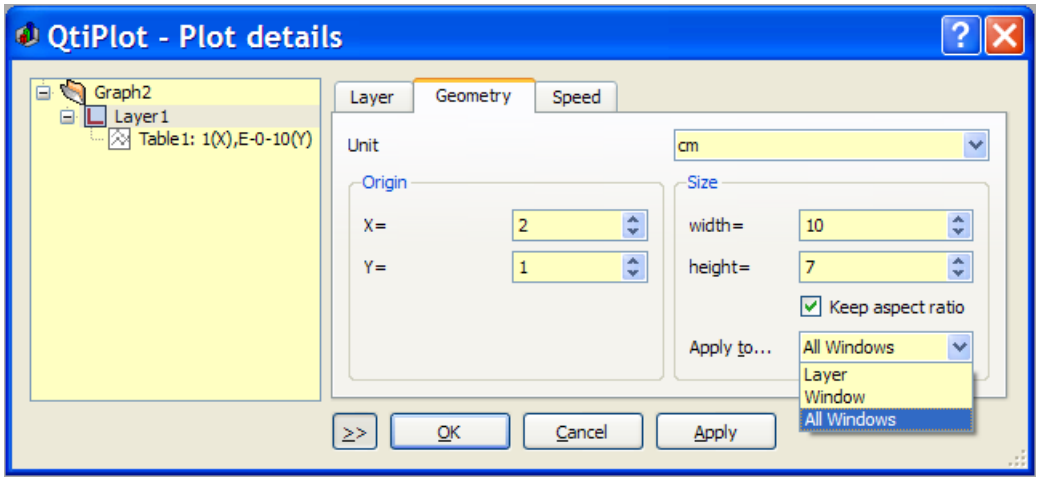


Figure 5.21: The Plot Details Dialog: Layer geometry.

The third tab can be used to customize the speed mode for the layer. This can be very usefull when dealing with very large data sets. The speed mode uses the Douglas and Peucker algorithm which purpose is that given a 'curve' composed of line segments to find a curve not too dissimilar but that has fewer points. The algorithm defines 'too dissimilar' based on the maximum distance (tolerance) between the original curve and the smoothed curve. The smoothed curve consists of a subset of the points that defined

the original curve. If the speed mode is enabled, the filtering of the data points is activated only for the curves with more than the maximum specified number of points.

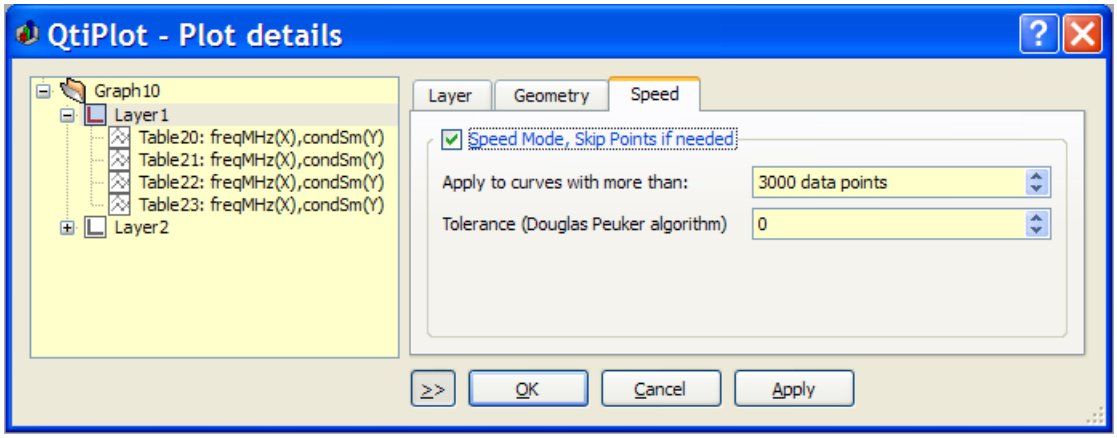


Figure 5.22: The Plot Details Dialog: Layer Speed Mode.

The right part of the dialog box contains several tabs which depend on the kind of plot that you are using, they are described in the following subsections. 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. 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.

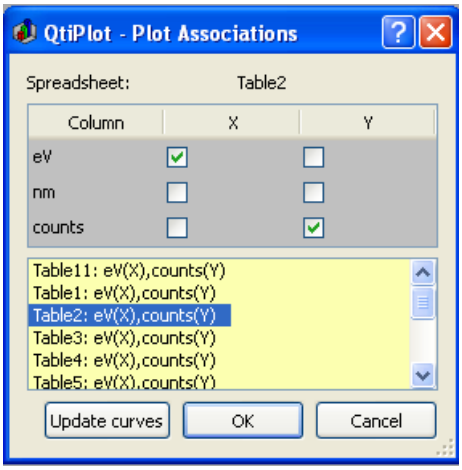


Figure 5.23: The Plot Details Dialog: Plot Associations.

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

5.10.1 Custom curves for lines and scatter plots

The first tab of the right part of the dialog window allows to define the pair of axes to which the curve is attached. You can thus represent different curves having different x/y scales on the same plot layer.

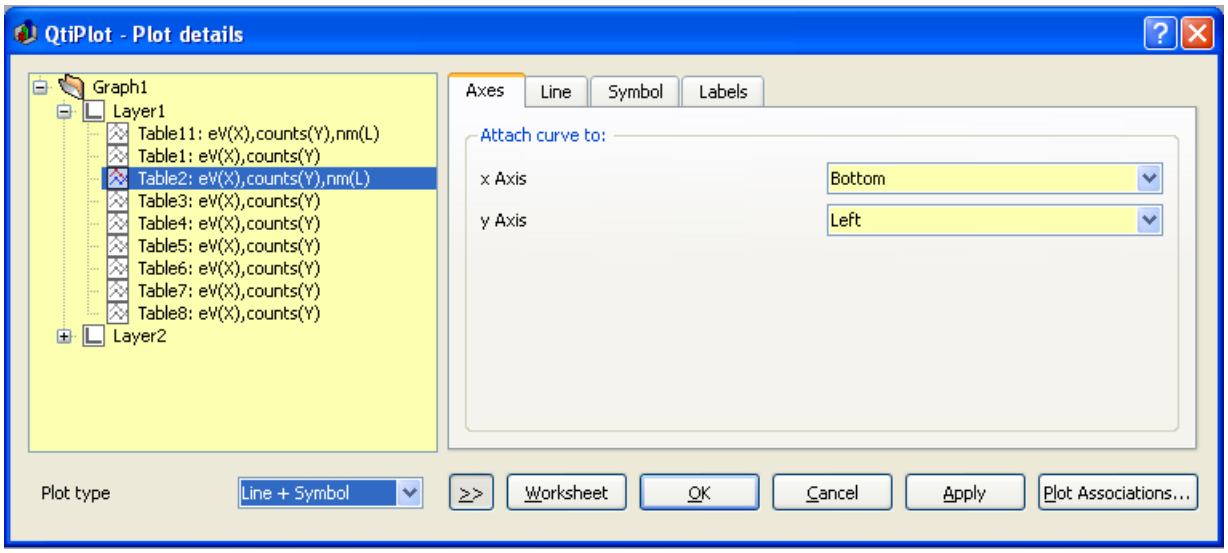


Figure 5.24: The Plot Details Dialog: assign axes.

The second tab of the right part of the dialog window 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 the different types of plot available.

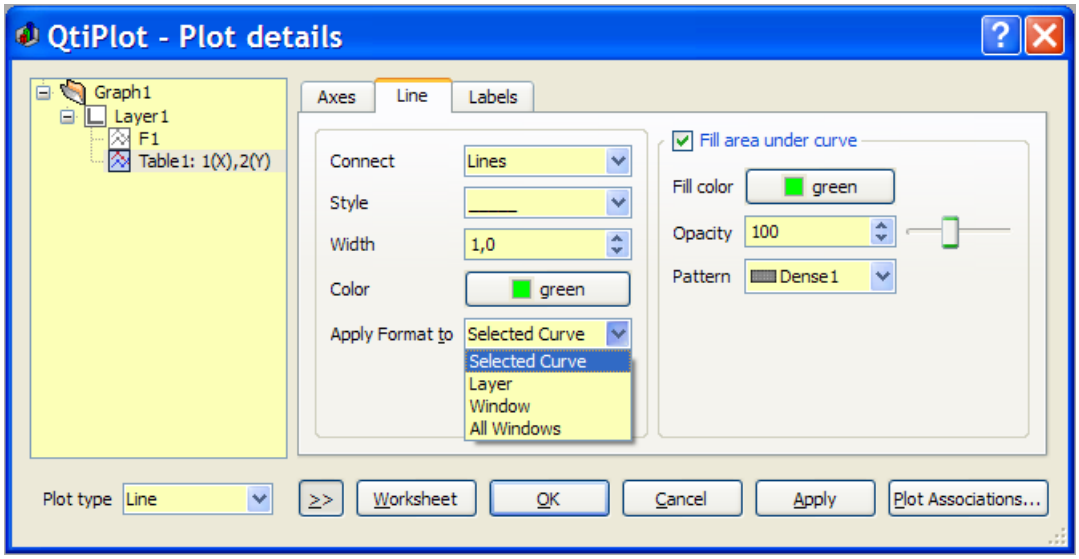


Figure 5.25: The Plot Details Dialog: Line formatting.

A third tab can be activated to select the symbol, and to modify the size, the color and the filling color of the symbols.

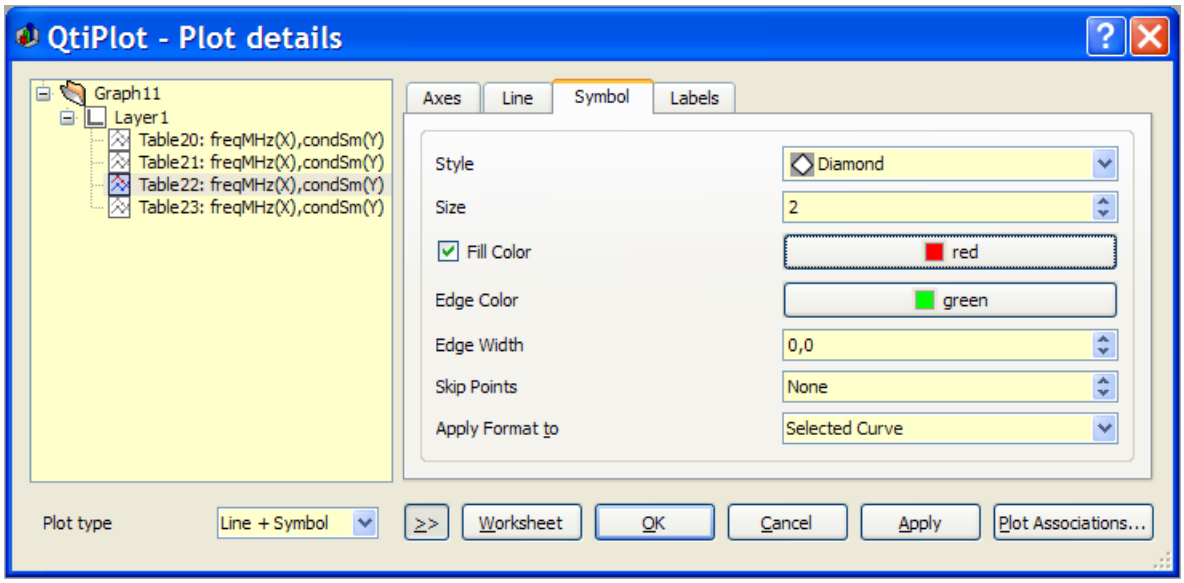


Figure 5.26: The Plot Details Dialog: Symbol formatting.

The *Labels* tab can be used to define labels to be plotted for each data point in the plot and to modify the font, the color, the rotation angle and the position of the labels.

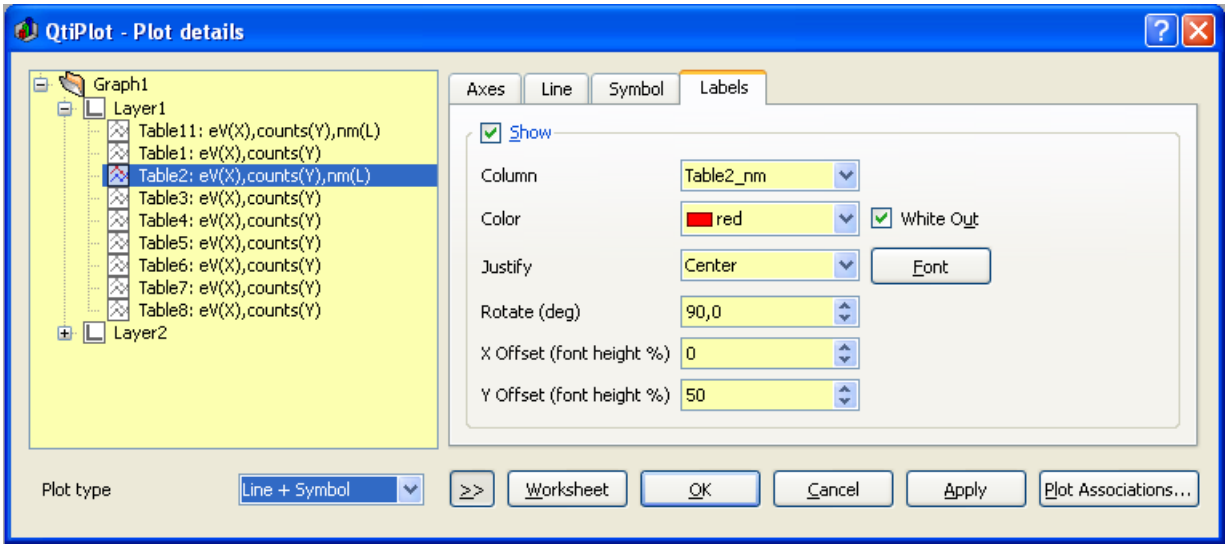


Figure 5.27: The Plot Details Dialog: Labels formatting.

5.10.2 Custom error bars

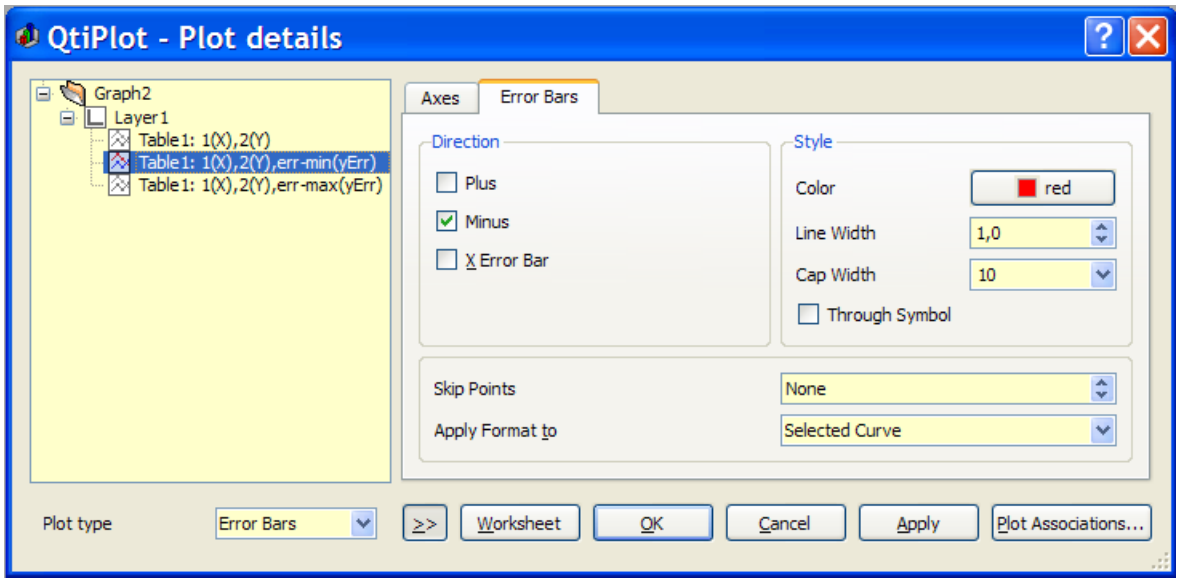


Figure 5.28: The Plot Details Dialog for error bars formatting.

5.10.3 Plot Details for pie plots

These commands are available for pie plots. 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.

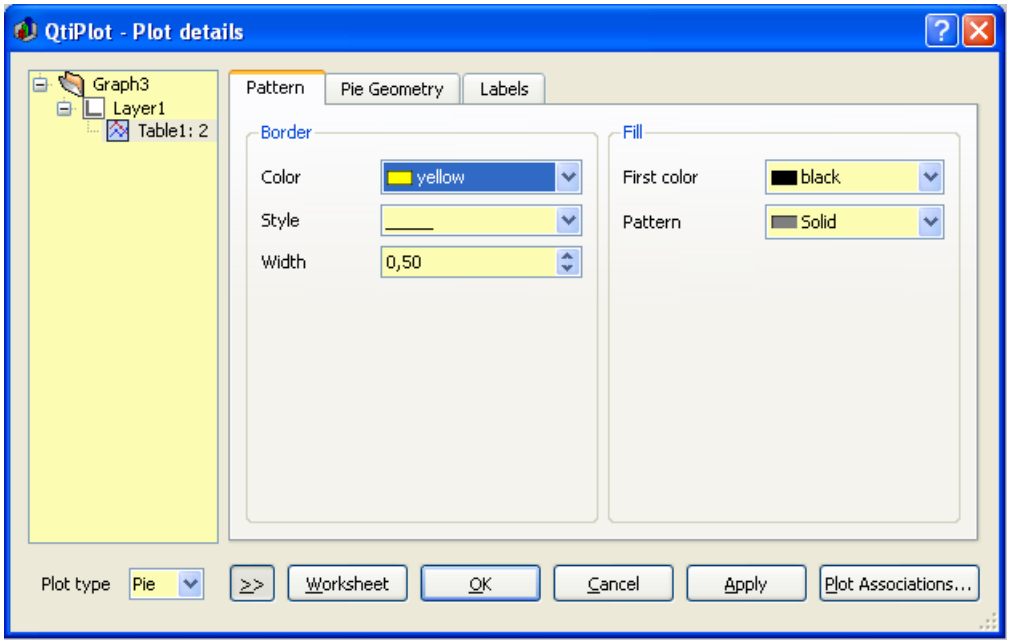


Figure 5.29: The Plot Details Dialog for pies: pie segment formatting.

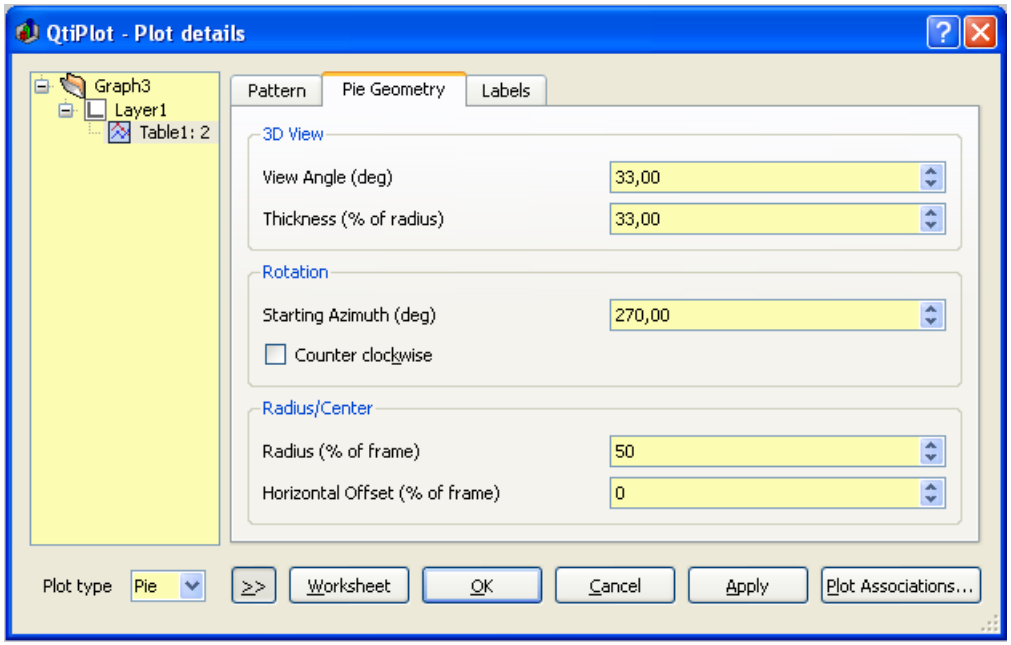


Figure 5.30: The Plot Details Dialog for pies: pie geometry.

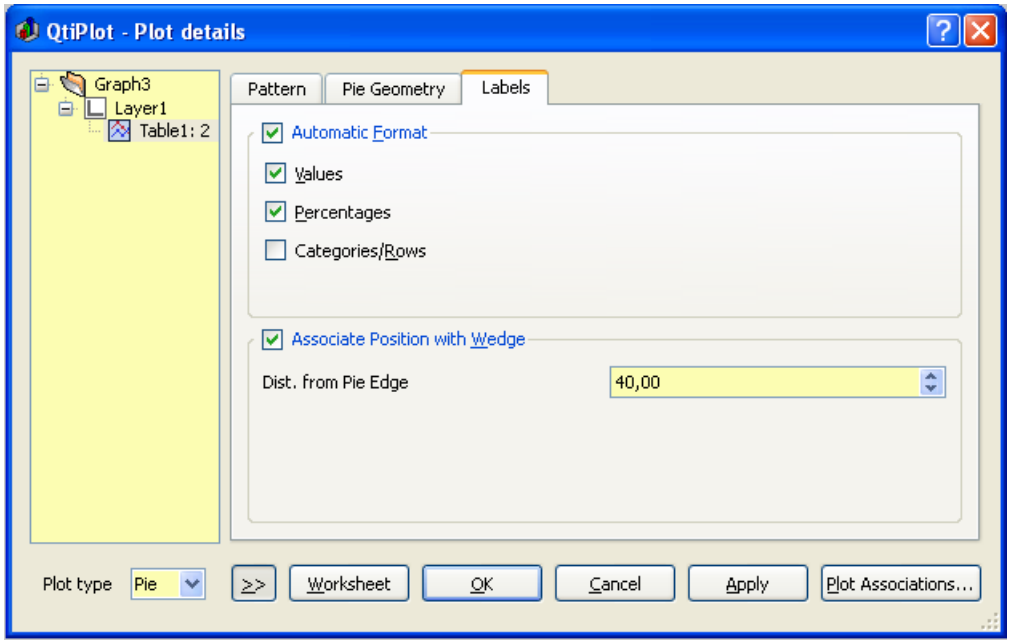


Figure 5.31: The Plot Details Dialog for pies: pie labels formatting.

5.10.4 Custom curves for box plots

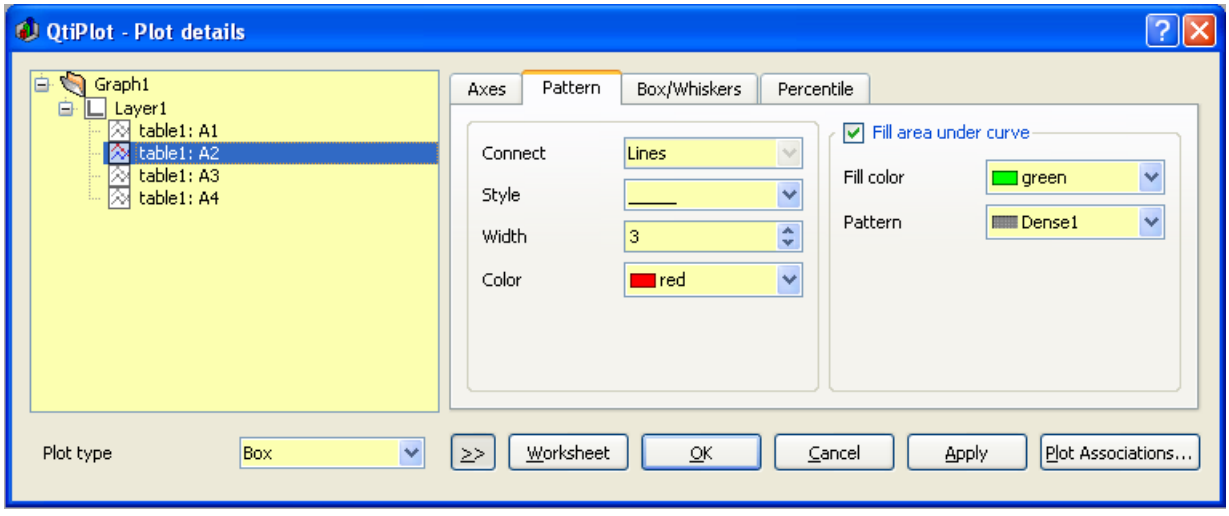


Figure 5.32: The Plot Details Dialog for box: pattern formatting.

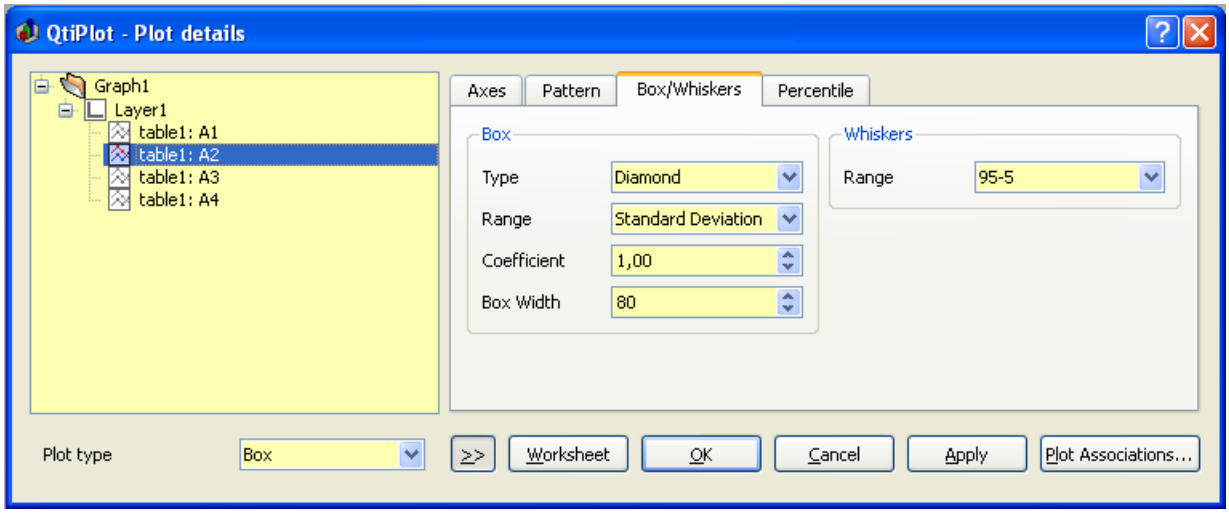


Figure 5.33: The Plot Details Dialog for box: whiskers formatting.

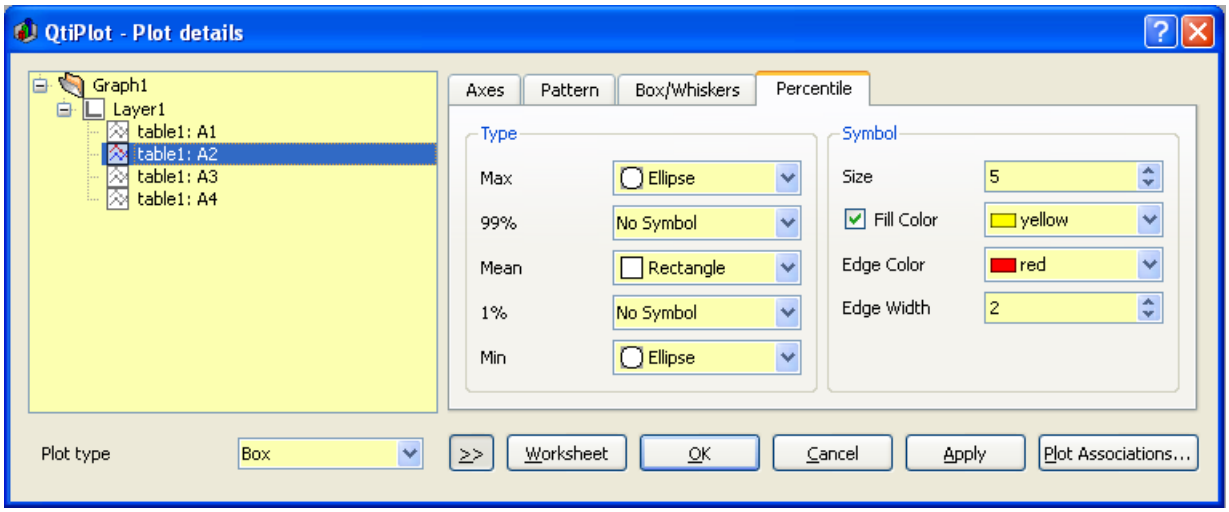


Figure 5.34: The Plot Details Dialog for box: percentile formatting.

5.10.5 Custom curves for pie histogram

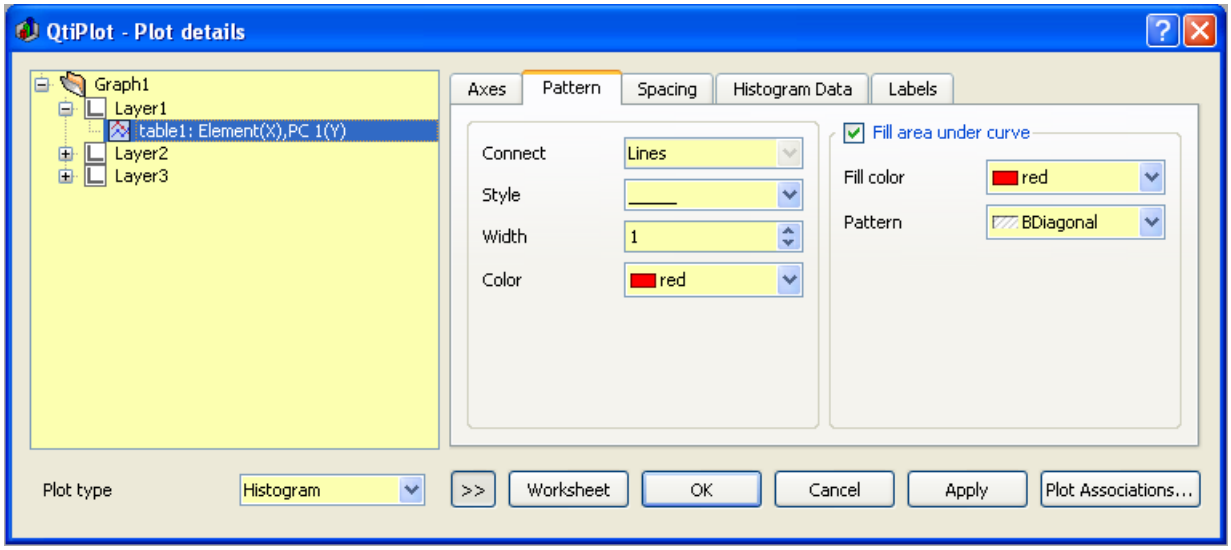


Figure 5.35: The Plot Details Dialog for histogram: pattern formatting.

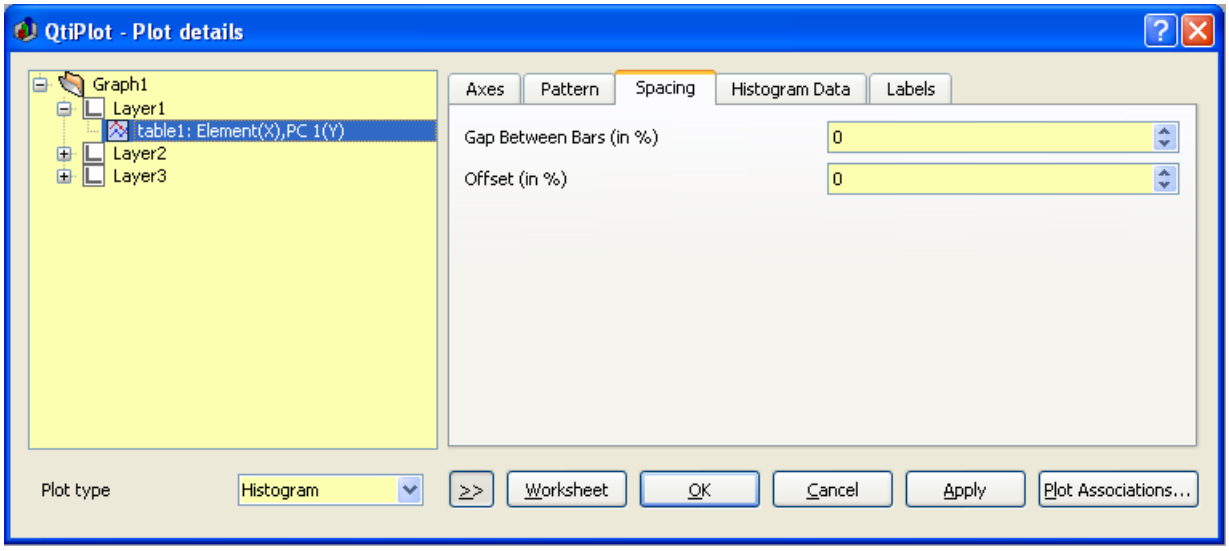


Figure 5.36: The Plot Details Dialog for histogram: spacing formatting.

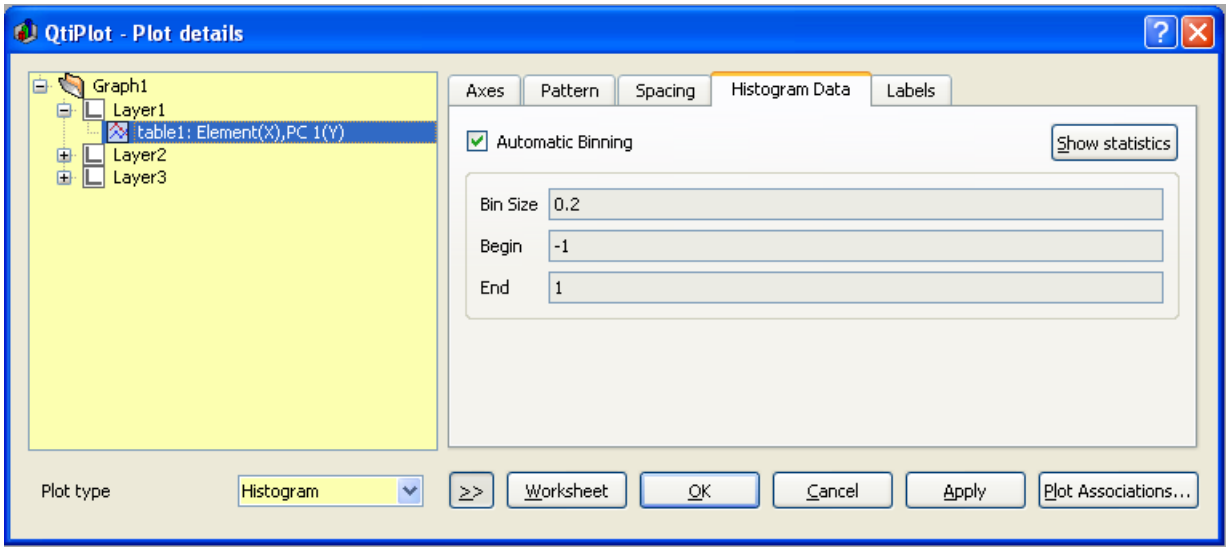


Figure 5.37: The Plot Details Dialog for histogram: data formatting.

5.11 Define surface plot

This dialog is used when you enter the **New -> New Surface 3D Plot** command. It allows to create a new function of two variables. The only available coordinate system is the cartesian one: z = f(x,y).

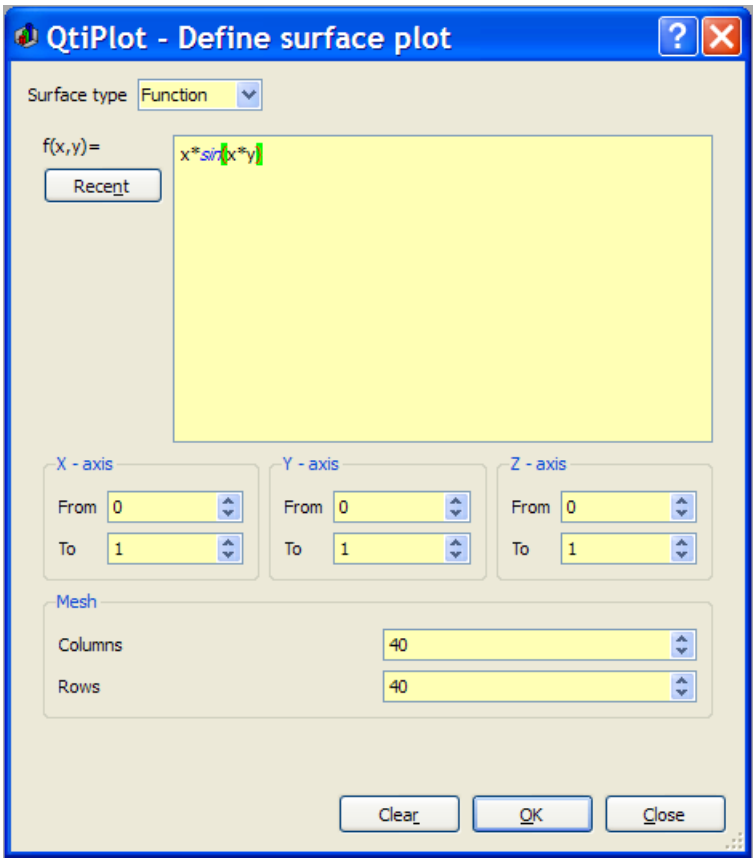


Figure 5.38: The **New -> New Surface 3D Plot** dialog box.

You can then enter the X, Y and Z scales and you can define the mesh parameters.

You can also create parametric surfaces. The only parameters allowed are the latitude and the longitude: `u` and `v`. Here's, for example, how you can plot a sphere having a radius equal to one:

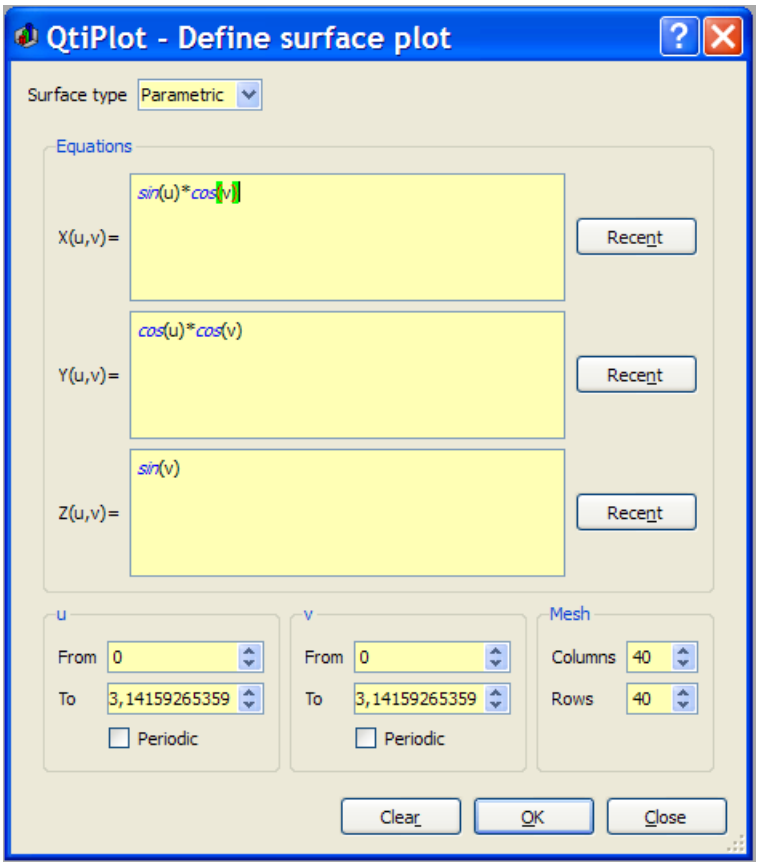


Figure 5.39: The **New -> New Surface 3D Plot** dialog box.

Like in the case of functions, you can supply the drawing domain for the two angle parameters and you can define the mesh parameters. The more rows/columns you request for, the better the resolution of the output image, but also the larger the memory consumption and the lower the speed of your computer.

Also, you can provide information about the rotational symmetry of the parametric surface to the drawing routines, using the two `Periodic` box options (one for each parameter).

5.12 Export ASCII

This dialog is activated by selecting the command Export ASCII from the File Menu. The command is active if there is at least a table window in the project.

This command is used to export all or a part of the data of a project to an ASCII file. Alternatively you can also choose to export the tables/matrices to one of the following

formats: TeX (.tex), Open Document Format (.odf) or as web pages (.html).

In this example of export of a table selection to an ASCII file, the names of the selected columns are exported as the first line of the file. The comments displayed in the table header are exported as the second line of the file and a TAB character is used as a separator between the columns. Only the two selected columns are exported.

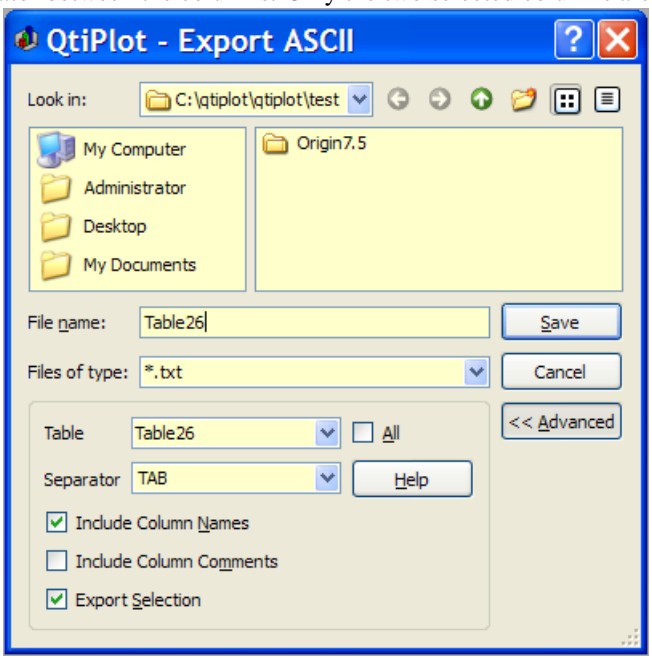


Figure 5.40: Export of a selection in a table to an ASCII file.

If the "Include Column Names" option is not checked, the names of the columns will be set to C1, C2,... in the exported file. The formatting of the numbers is kept in the ASCII file, so you have to be careful to obtain a good enough precision in the ASCII file.

5.13 Fast Fourier Transform

The **FFT...** dialog box can be used either on a table or on a plot. It is used to compute a direct or inverse FFT. See the FFT section in the Analysis chapter for an example.

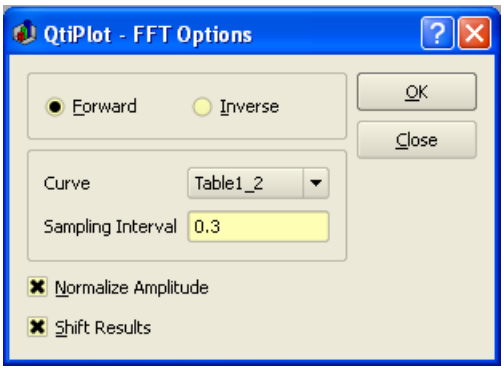


Figure 5.41: The **FFT...** dialog box for a curve.

QtiPlot 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.

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.

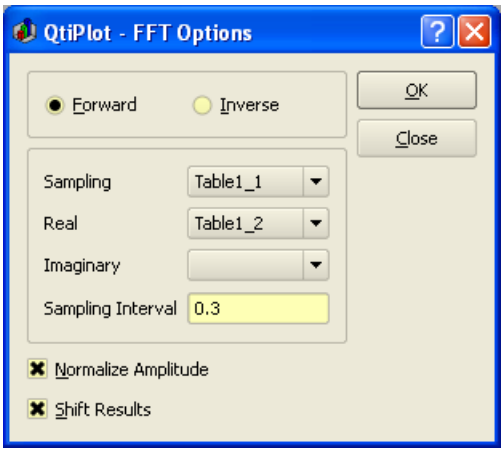


Figure 5.42: The **FFT...** dialog box for a table.

In the case of a table, you must select the sampling column (X-values) and two columns for Y-values. If they are complex numbers, the first column is the real part of Y-values and the second is the imaginary part. If Y-values are simple reals, you don't have to select a column for the imaginary part, you can leave the combo box empty.

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

5.14 Integrate dialog

This dialog box is opened if you select the **Integrate Function...** command from the Analysis menu The *Function* text box allows you to edit the analytical function that will be integrated. The second input field displays the name of the independent variable of the function and is set to *x* by default. The third parameter is 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.

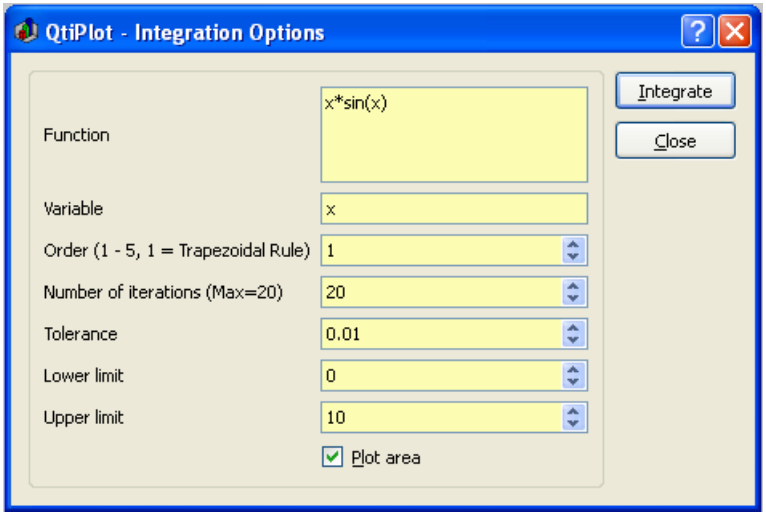


Figure 5.43: The **Integrate...** dialog box.

The result of the integration will be given in the The Project Explorer.

5.15 The Fit Wizard

This dialog is activated by selecting the command Fit Wizard... from the Analysis Menu. This command is active if a plot or a 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 dialog 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 availables: the user defined functions which have been saved, the classical functions proposed by QtiPlot 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.

This first step is used to define the function which will be used for the fitting

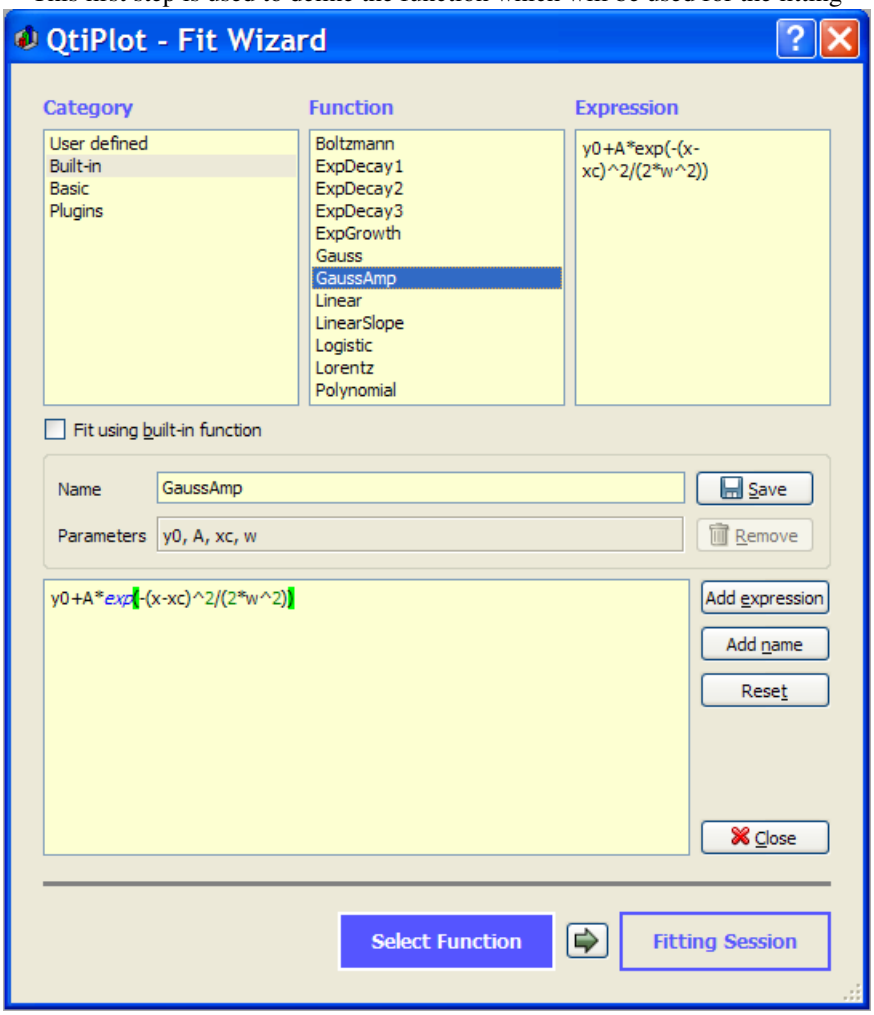


Figure 5.44: The first step of the **Fit Wizard...** dialog box.

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

This second step is used to define the parameters of the fitting

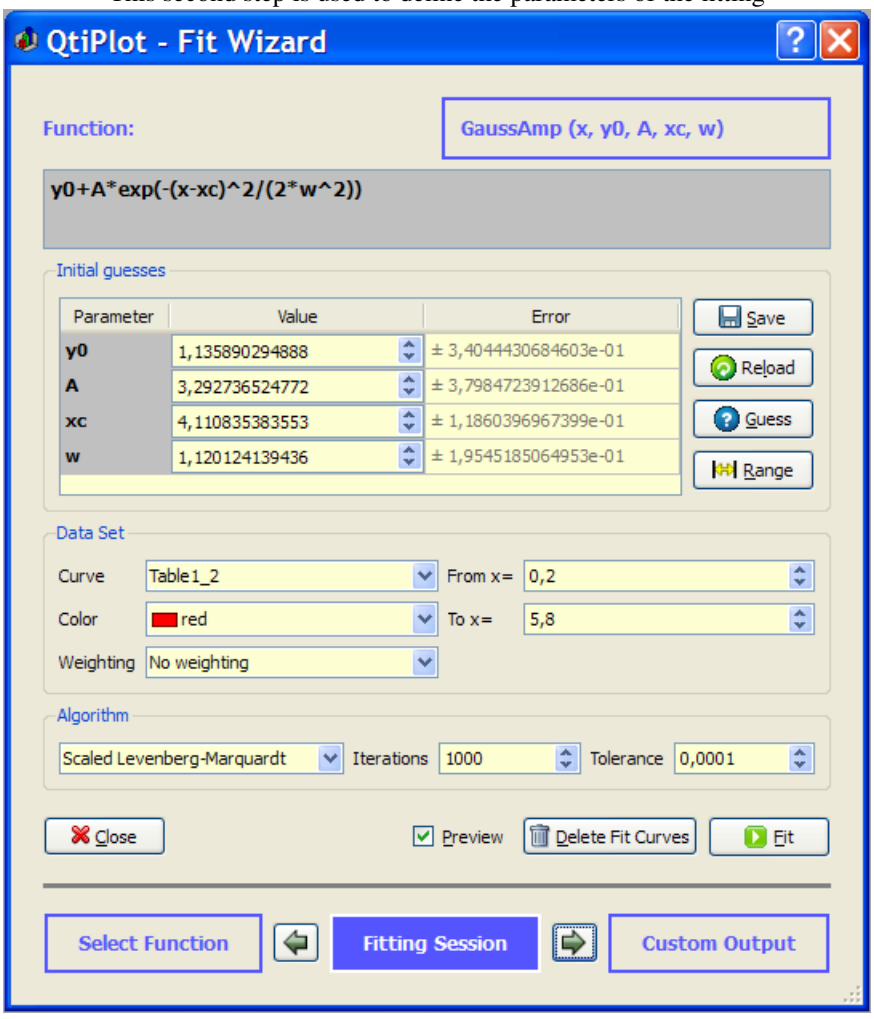


Figure 5.45: The second step of the **Fit Wizard...** dialog box.

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. *No weight*: all weighting coeficients are set to 1 ($w_i = 1$).
2. *Instrumental*: the values of the associated error bars are used as weighting coeficients $w_i = 1/er_i^2$, where er_i are the error bar sizes stored in error bar columns. You must add Y-error bars to the analysed curve before performing the fit.
3. *Statistical*: the weighting coeficients are calculated as $w_i = 1/y_i$, where y_i are the y values in the fitted data set.

4. *Arbitrary Dataset*: you have the possibility to set the weighting coeficients using an arbitrary data set $w_i = 1/c_i^2$, where c_i are the values in the 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.

5. *Direct Weighting*: you have the possibility to set the weighting coeficients using an arbitrary data set $w_i = c_i$, where c_i are the values in the 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.

This third step is used to customize the output of a fitting operation.

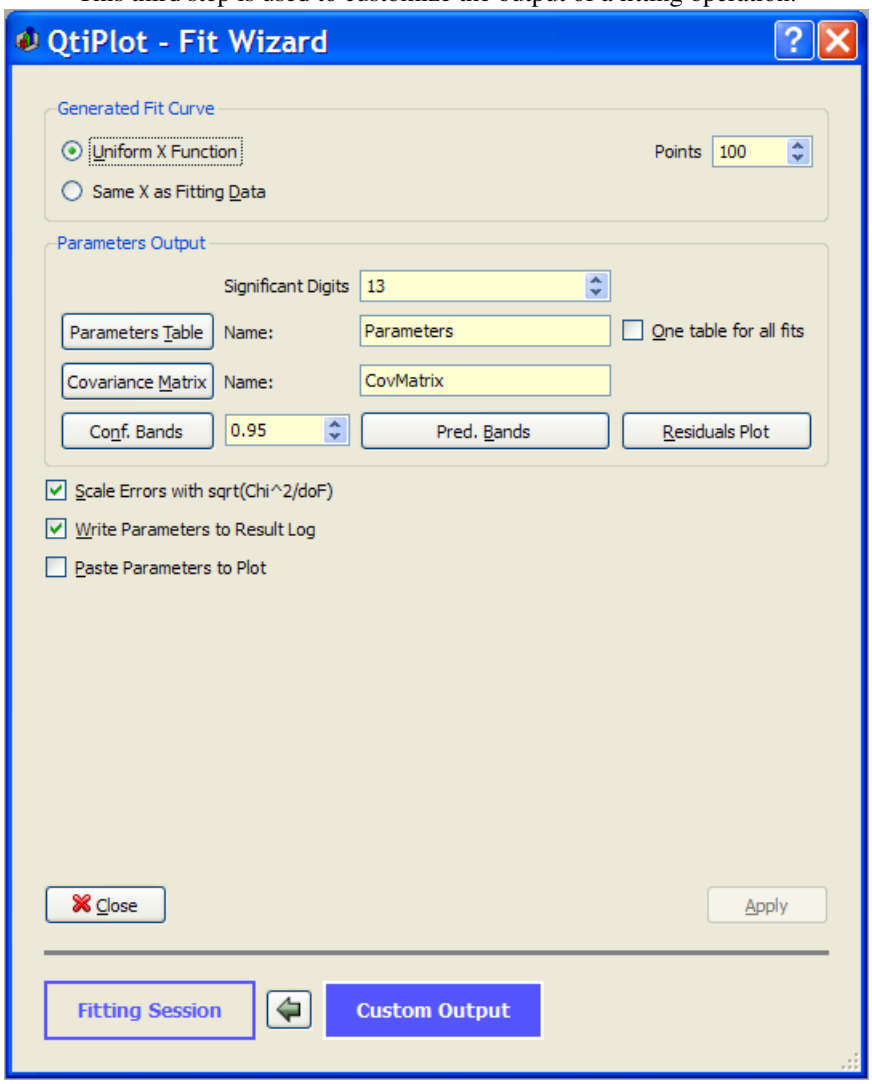


Figure 5.46: The third step of the **Fit Wizard...** dialog box.

This dialog tab provides controls that can be used to evaluate the goodness of fit. The first one is the *Residuals Plot* button which can be used to display the curve of the plot residuals. The *Conf. Bands* and *Pred. Bands* buttons can be used to generate confidence and prediction limits for the current fit, based on the user input confidence value.

After the fit, a series of fit statistics are displayed in the log window allowing to evaluate the goodness of fit. This values are:

1. Chi^2/doF
2. R-square
3. Adjusted R-square
4. RMSE (Root Mean Squared Error)
5. RSS (Residual Sum of Squares)

For detailed explanations about the meaning of this statistical values, please visit the following link.

By default, the reported errors are not automatically scaled with the square root of the reduced chi-squared value. You can choose to enable this option by checking the *Scale errors with sqrt(Chi^2/doF)* box.

5.16 General Plot Options

The first tab is used to set the general scales used for the two or three axis.

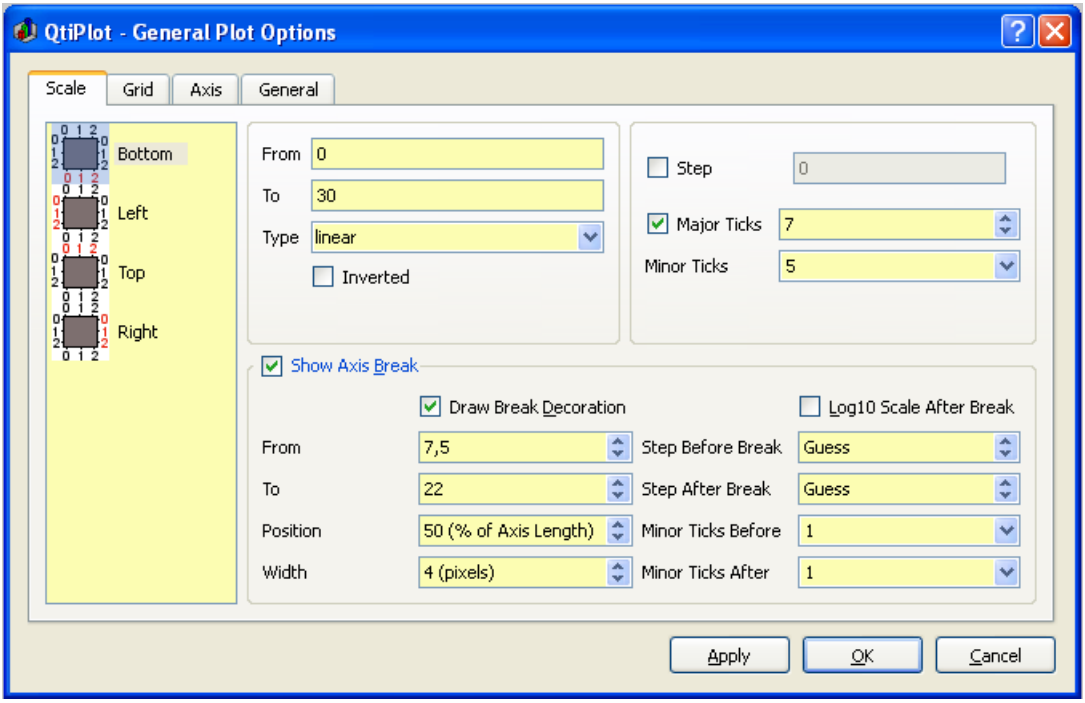


Figure 5.47: General plot options dialog: the scale tab.

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, QtiPlot will use a value which leads to a pretty plot: for example, if you enter 7 ticks for a 0..100 scale, QtiPlot will use 10 major ticks from 10 to 10. If you want to fix non classical values, you can select the step method.

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.

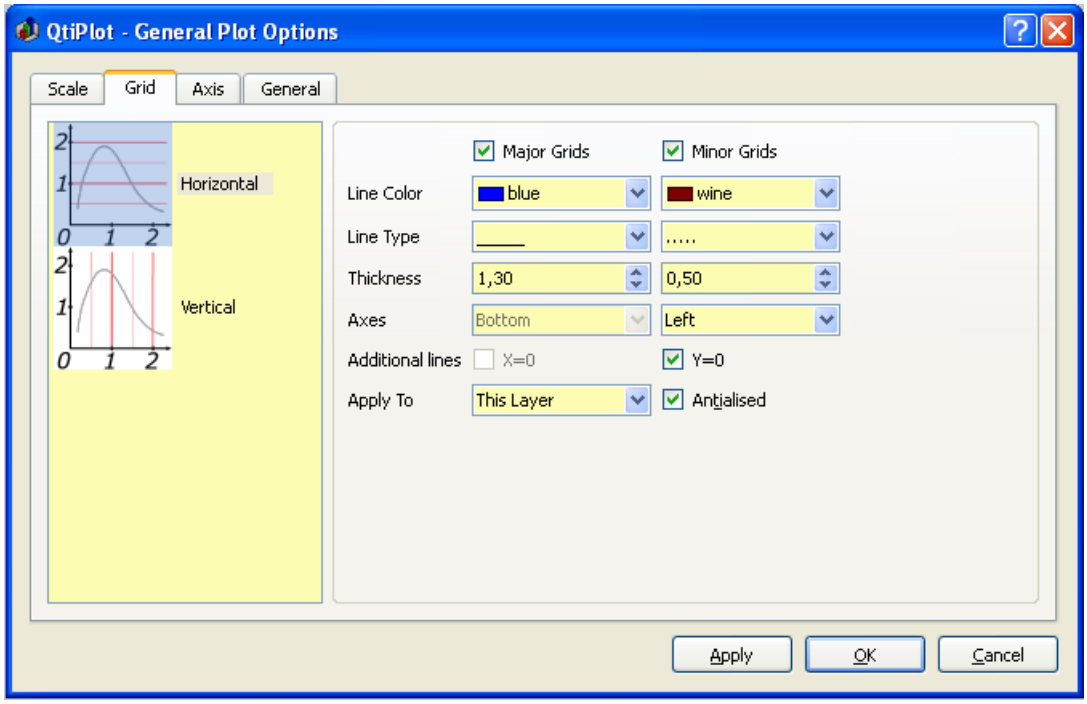


Figure 5.48: General plot options dialog: the grid tab.

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 options dialog section for more details.

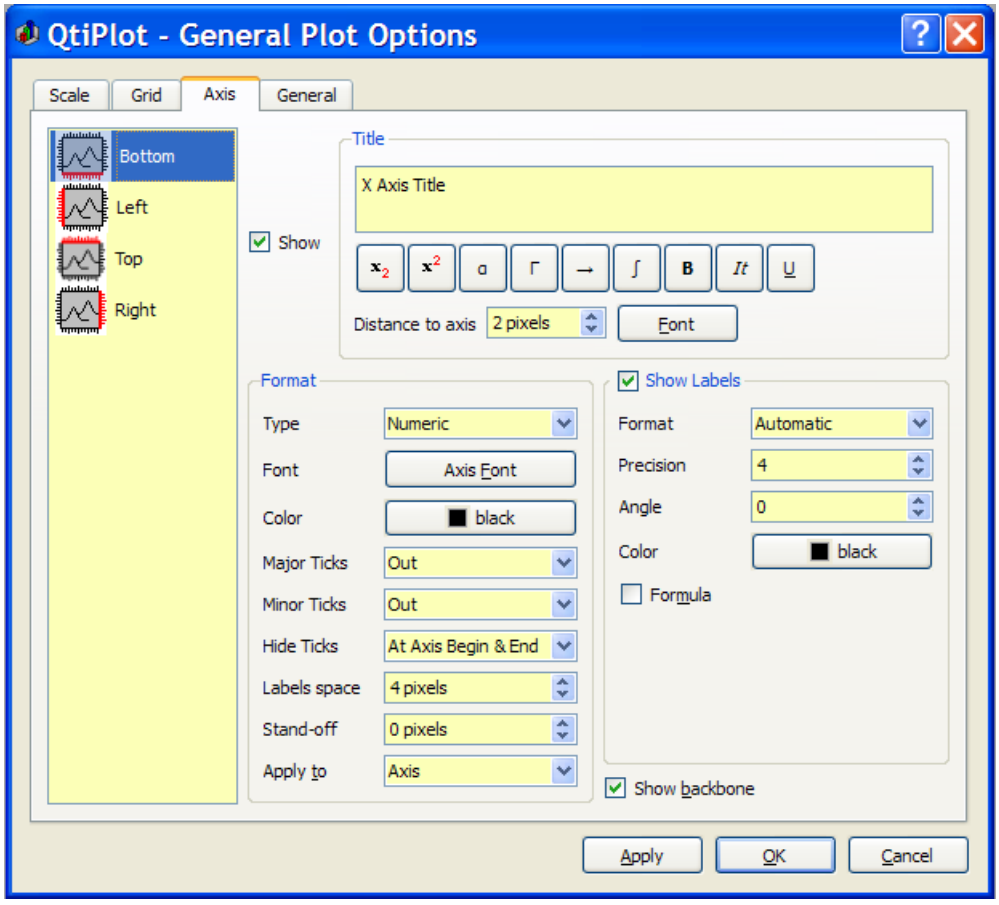


Figure 5.49: General plot options dialog: the axis tab.

The *General* settings tab is used to customize the global aspect of the plot. 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 dialog.

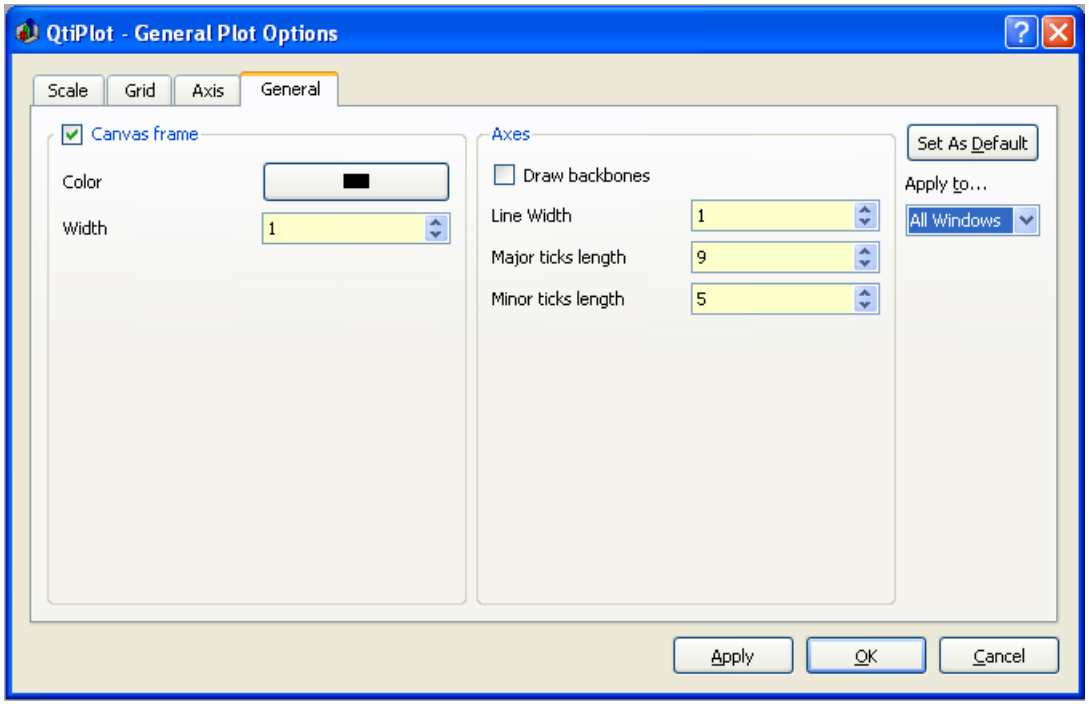


Figure 5.50: General plot options dialog: General settings.

The parameters in the *Axes* group allow to modify the linestyle of the axes and of the ticks.

5.17 Plot Wizard

This dialog is activated by selecting the command Plot Wizard from the View Menu or with the Ctrl-Alt-W key. This command is always active.

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 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)

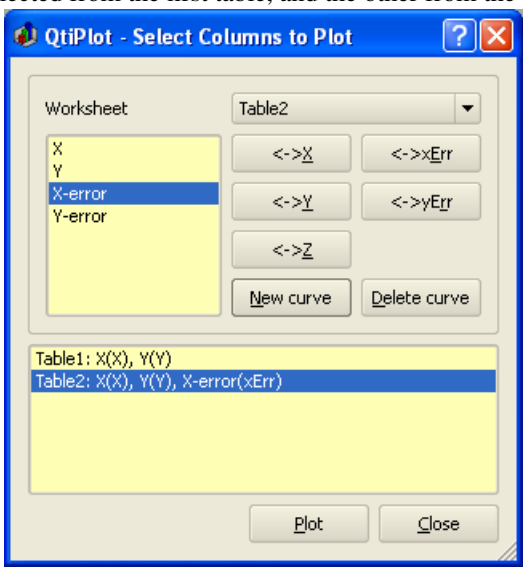


Figure 5.51: The plot wizard dialog box.

5.18 Project Explorer

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.

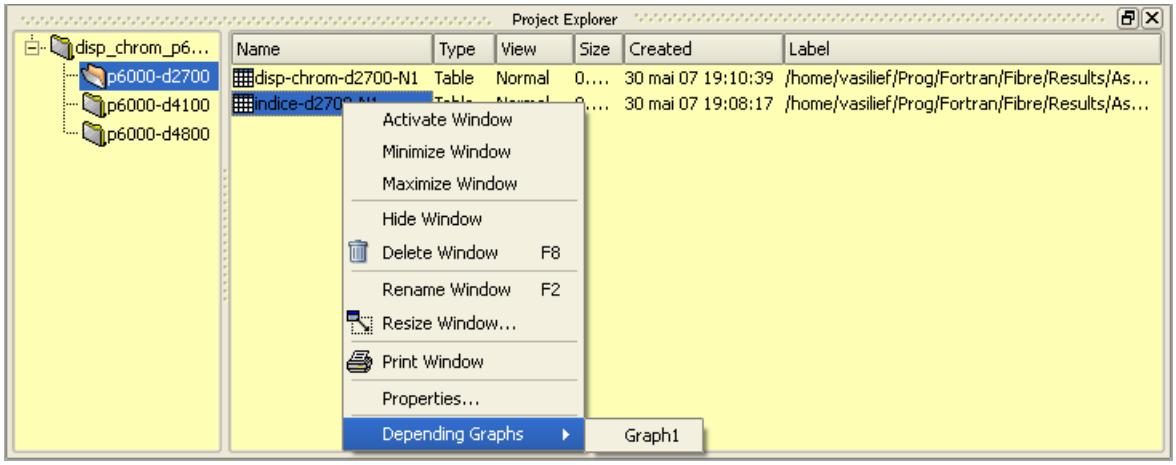


Figure 5.52: The project explorer panel.

5.19 Preferences Dialog

The preference dialog is used to customize the application. It has six 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. 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 translations available can be downloaded from the following address: http://soft.proindependent.com/translations.html. By default the translations must be place in a folder called *translations*, situated in the same location as the QtiPlot executable, in order to be loaded by the application, but you can specify a different folder via the *File Locations* tab.

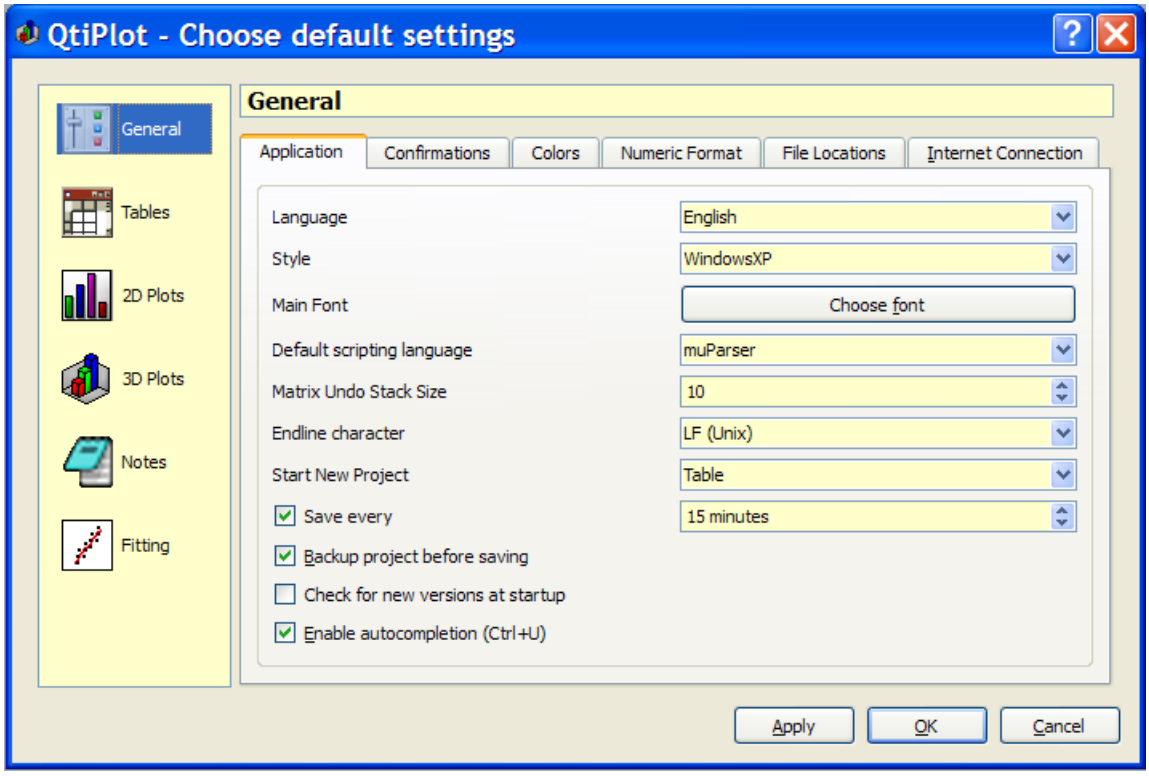


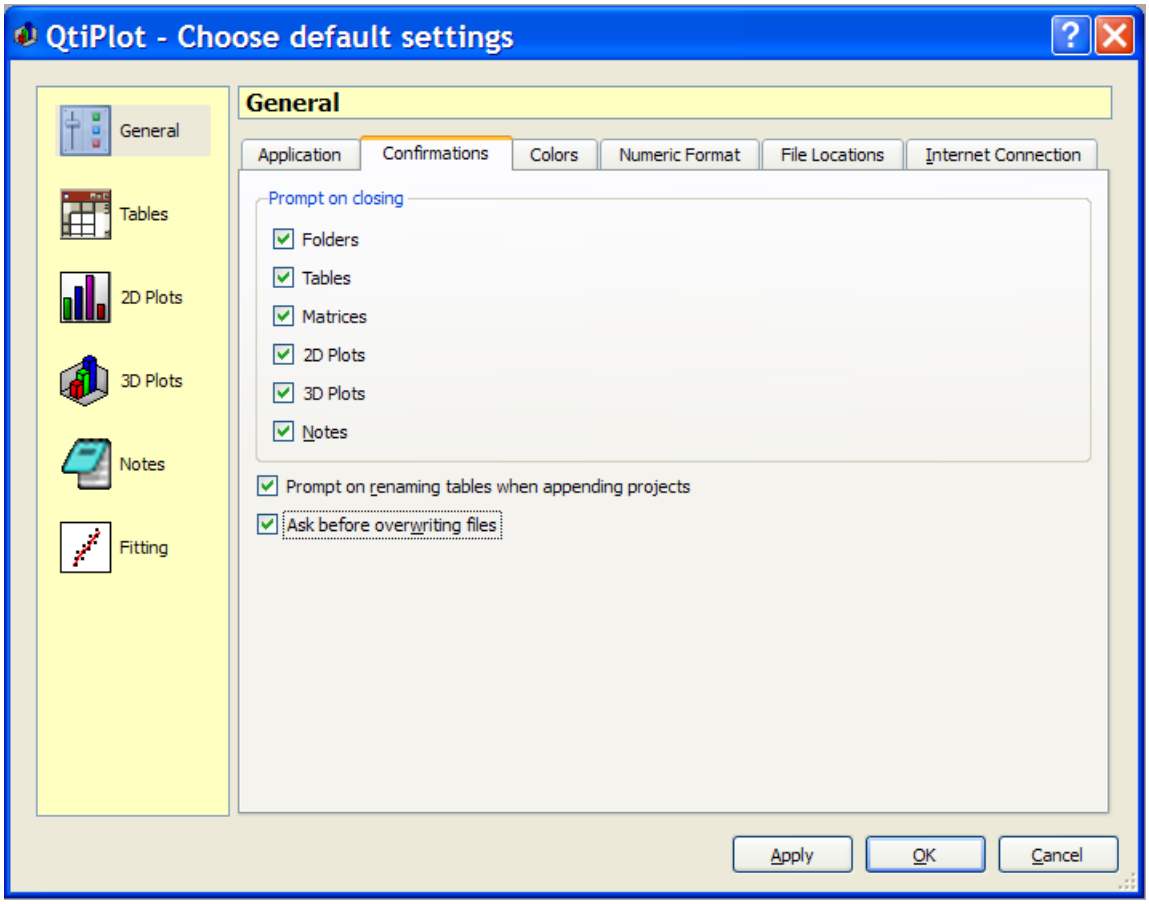
Figure 5.53: The preferences dialog: general parameters for the application.

The *Matrix Undo Stack Size* is the number of operations that can be undone/redone when working on a matrix. By default it is set to ten operations. A high value for this parameter can be very costly in terms of memory consumption, especially for large matrices.

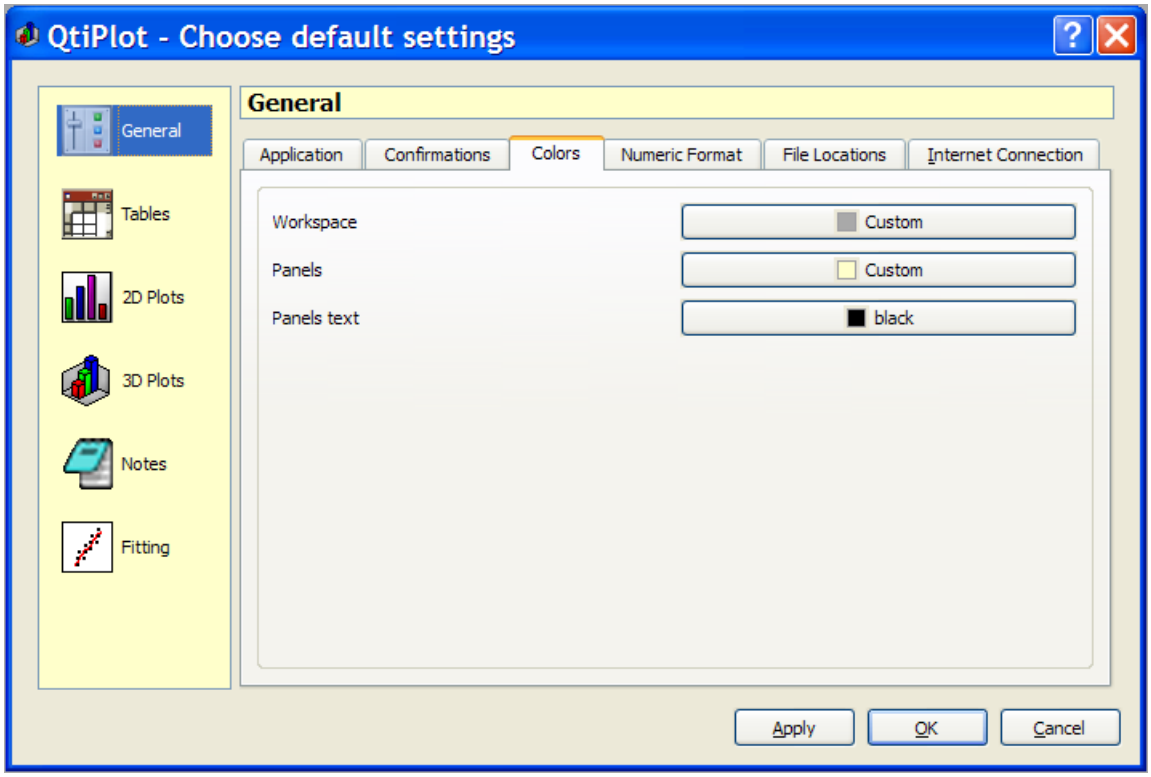
The *Endline character* defines the end of line convention used by QtiPlot for copy-/paste operations and for exporting matrices/tables to ASCII files. The end of line convention can be: *Line Feed (LF)*, *Carriage Return + Line Feed (CRLF)* or *Carriage Return (CR)* only.

Starting with version 0.9.6, autocompletion is enabled by default in all QtiPlot editors: in Notes, in the Script Window and in the values dialogs for matrices and tables. The autocompletion mechanism is based on a list of words provided by the *qti_wordlist.txt* file. This file, which is shipped with the source archive, must be placed in the same folder as the Python configuration files (see *File Locations* tab bellow), and is automatically loaded by QtiPlot on start-up. You can edit this file and add your own key words: one word per line. Completion suggestions are automatically poped-up for words that have more than two characters, but you can trigger autocompletion at any time using the shortcut Ctrl+U. Autocompletion can be disabled by unchecking the *Enable autocompletion* option.

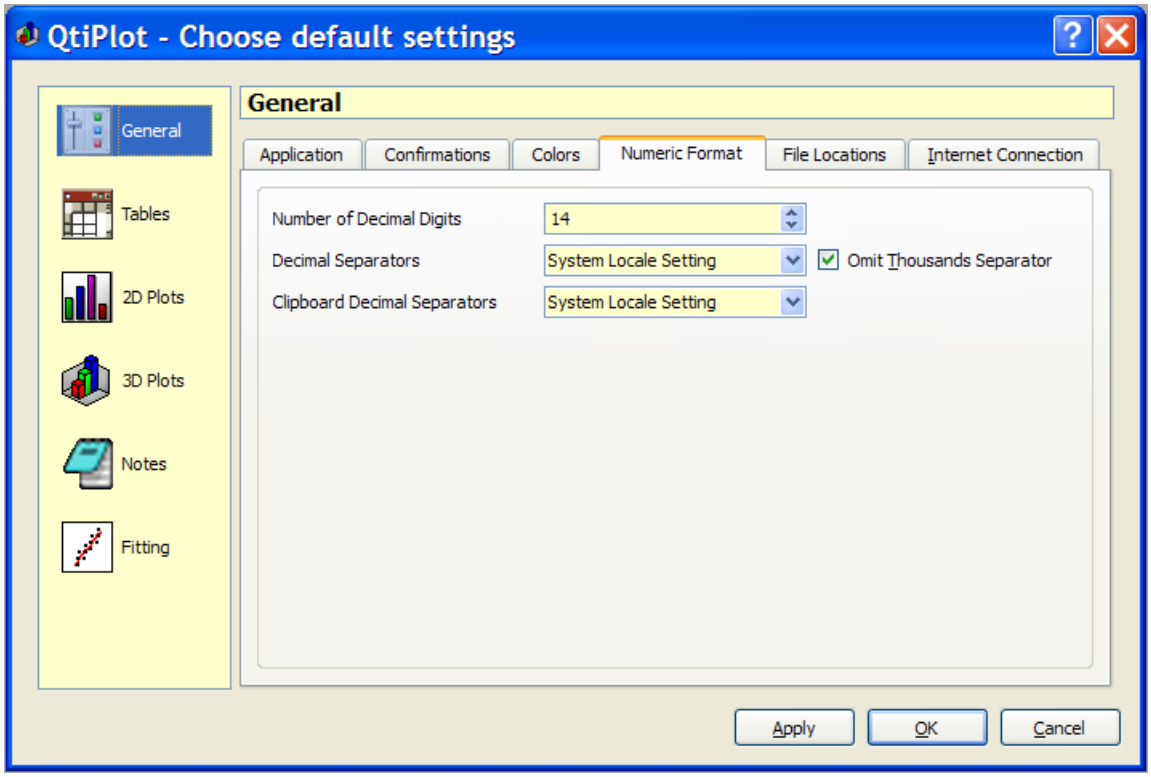
The second tab of the *General* option set is used to disable the prompting on deleting project windows. Also you can disable the warnings prompted by default when renaming new table windows with names already used in the current project.



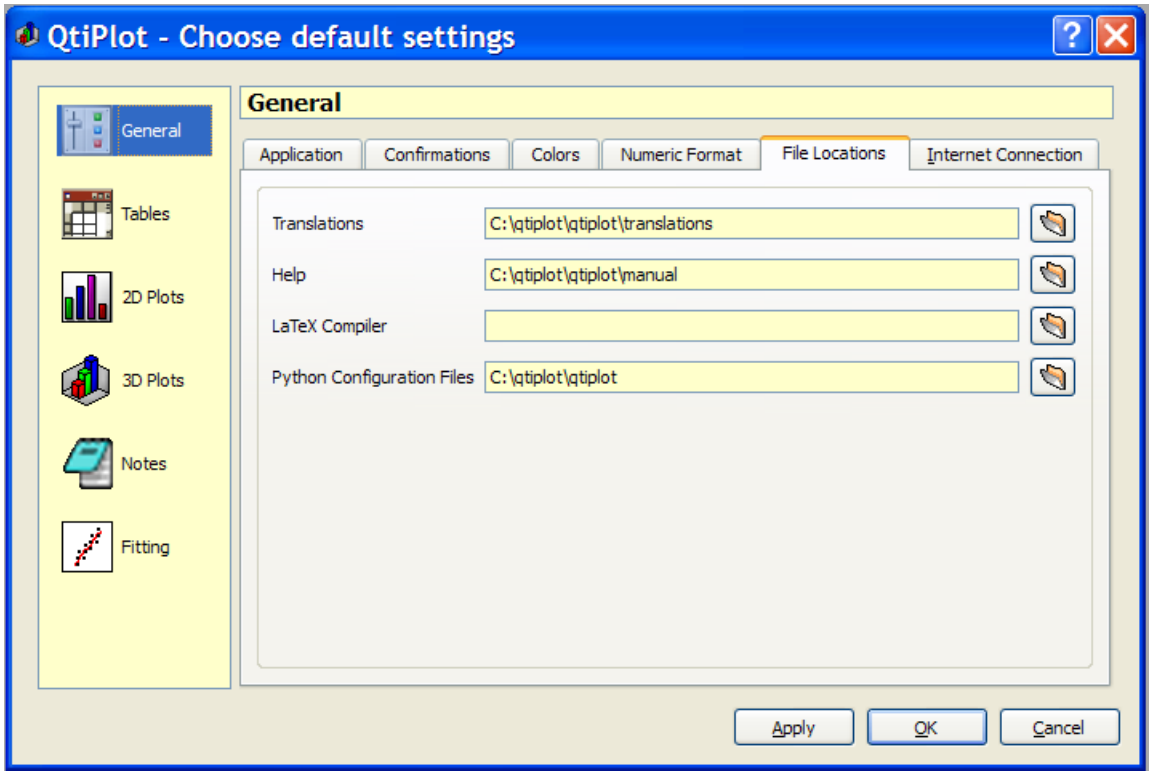
In this tab, you can change the default color for the workspace of the application. You can also choose the background color and the text color for panels. The panels are the Log Window and the Project explorer.



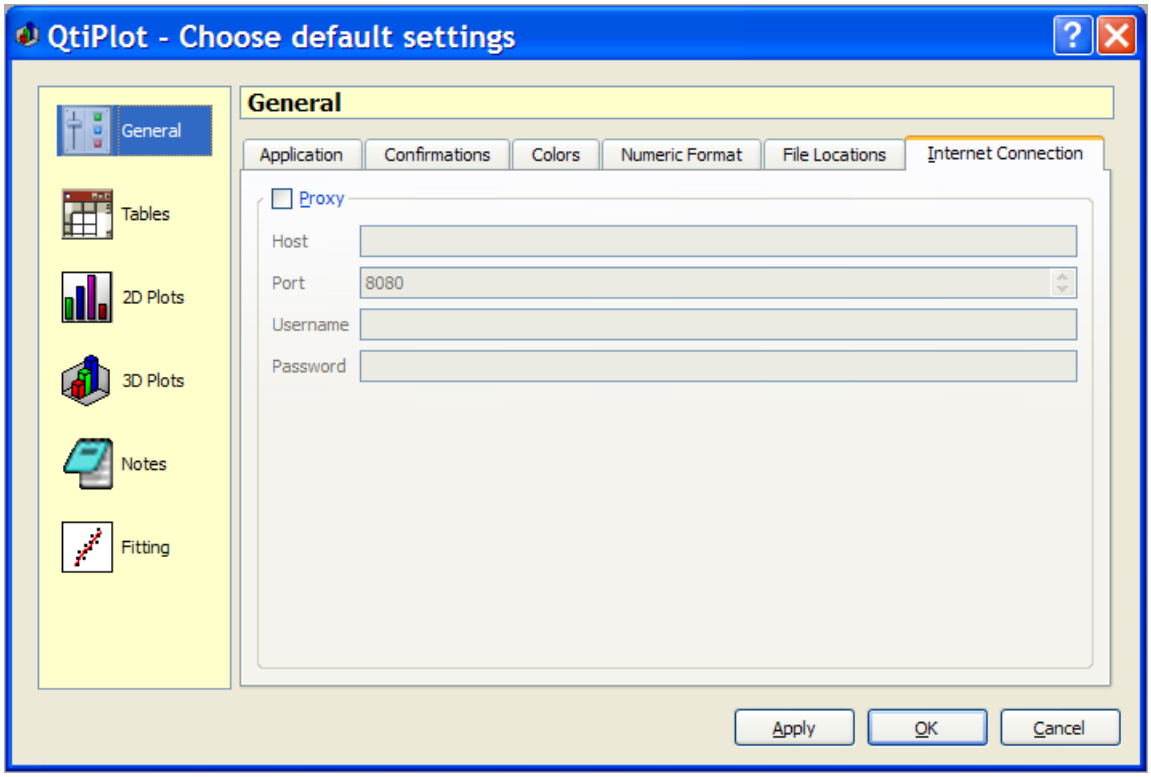
The *Numeric Format* tab allows you to customize the characters used as decimal point and as thousands separator. By default, QtiPlot uses the locale settings detected on your system. QtiPlot will convert all the existing data in your project to the new settings when you click the *Apply* button. The *Number of Decimal Digits* specifies the default precision used for any calculations operated on your data in Tables/Matrices.



The *File Locations* tab allows you to define custom locations for the folders containing the translation files, the manual files and the Python configuration files, *qtiplotrc.py* and *qtiUtil.py*, in case QtiPlot was built with Python scripting support.

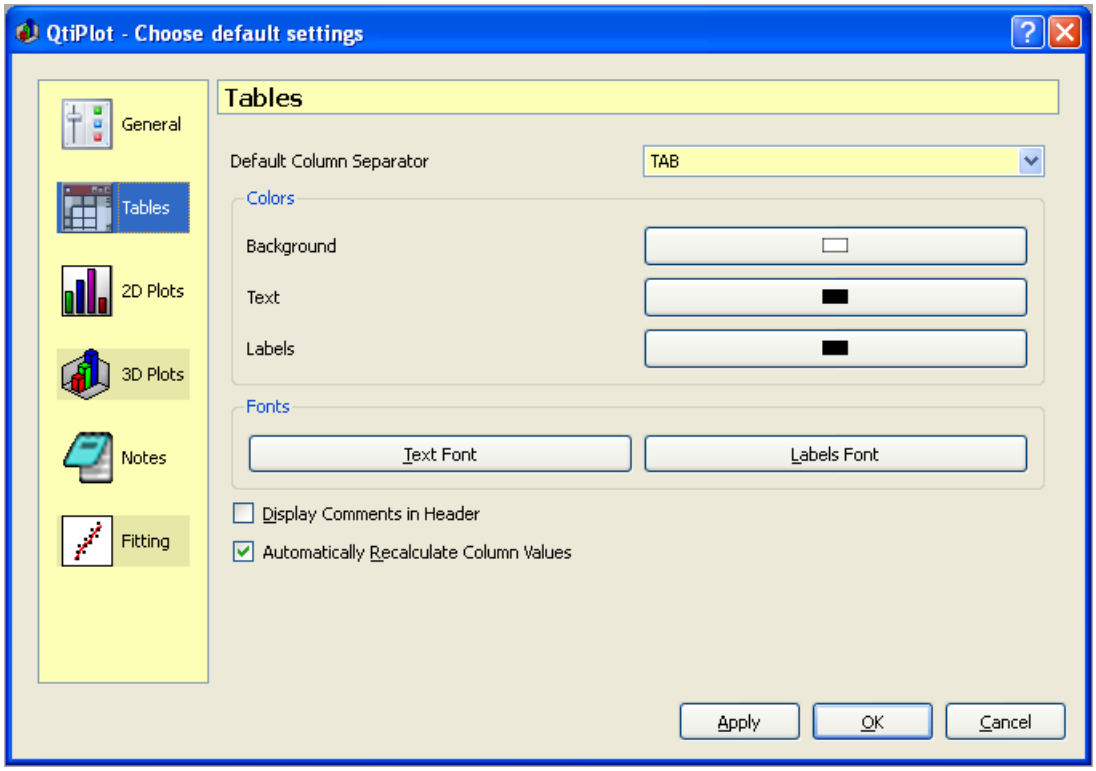


The *Internet Connection* tab is useful only in case you connect to the internet via a proxy.



The second set of option allows to customize the defaut aspect of tables: background, text colors, and fonts for tables and labels. By checking the *Display Comments in Header* option, the column comments will also be displayed in the table header, bellow the column names. If the *Automatically Recalculate Column Values* option is checked, all modifications in the values of a column trigger a recalculation of all columns with formulas depending on the modified column.

The preferences dialog: table options.



The second set of options is used to customize the default aspect of *2D plots*. The first tab is used to modify some general options. Most of the changes made to these options will be applied only to the newly created plots. Only a few of the changes, like *Autoscaling* of the plot axes, *Antialising* of curves and the behavior on resize events will also affect the already existing plots.

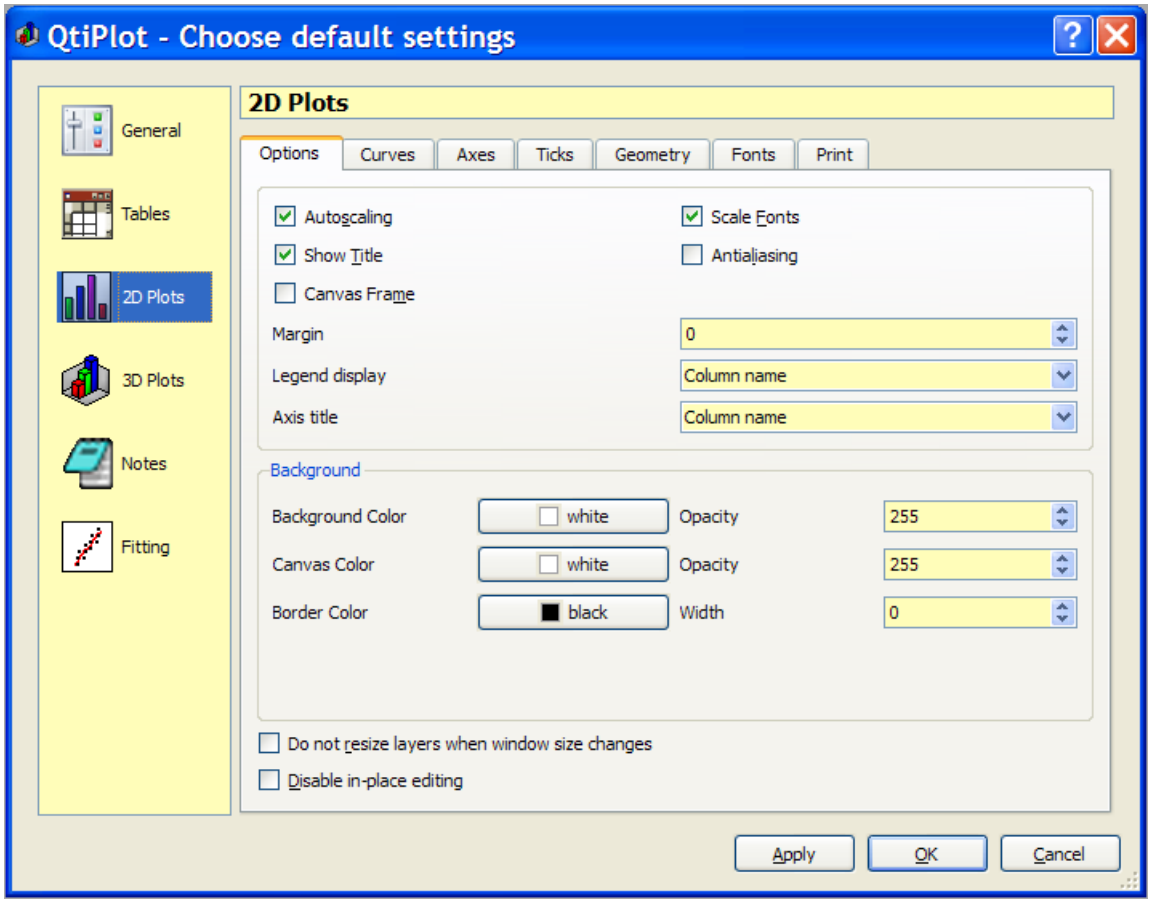
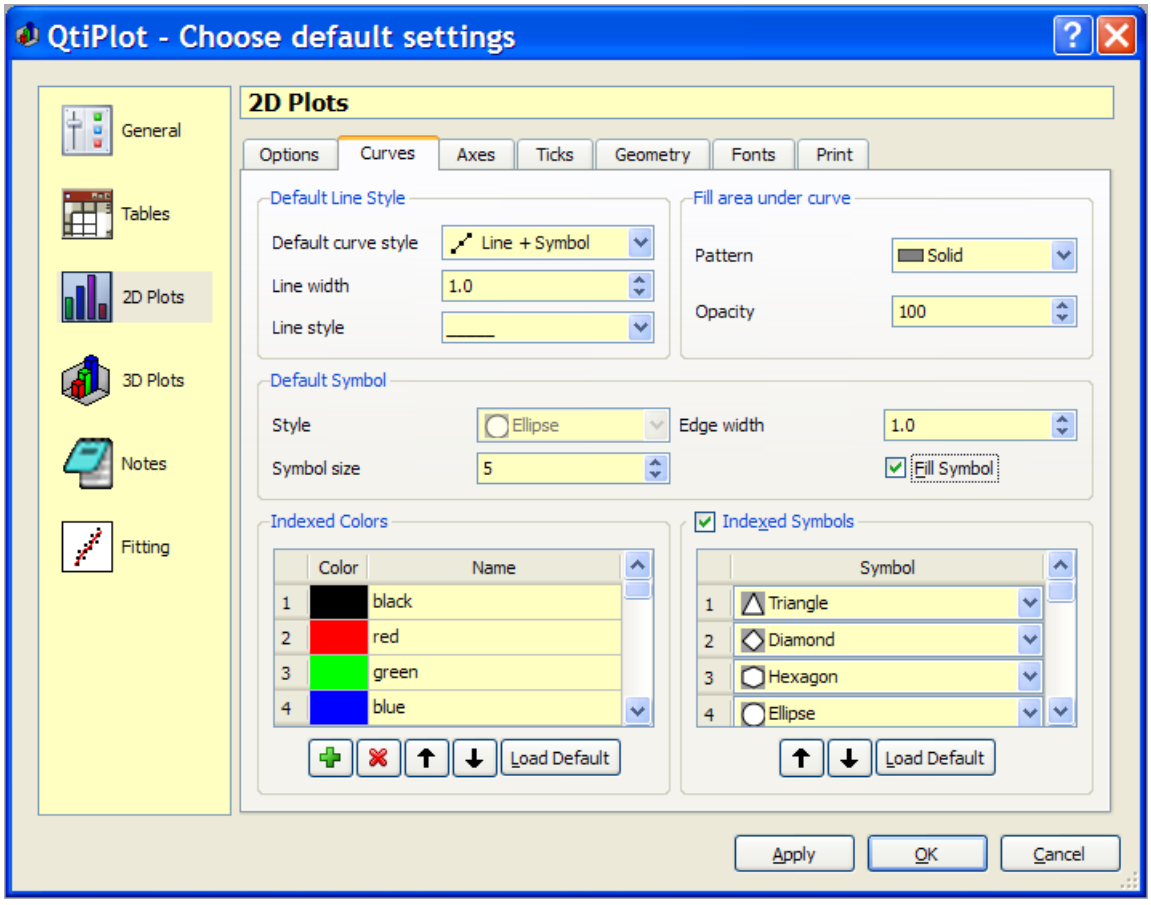
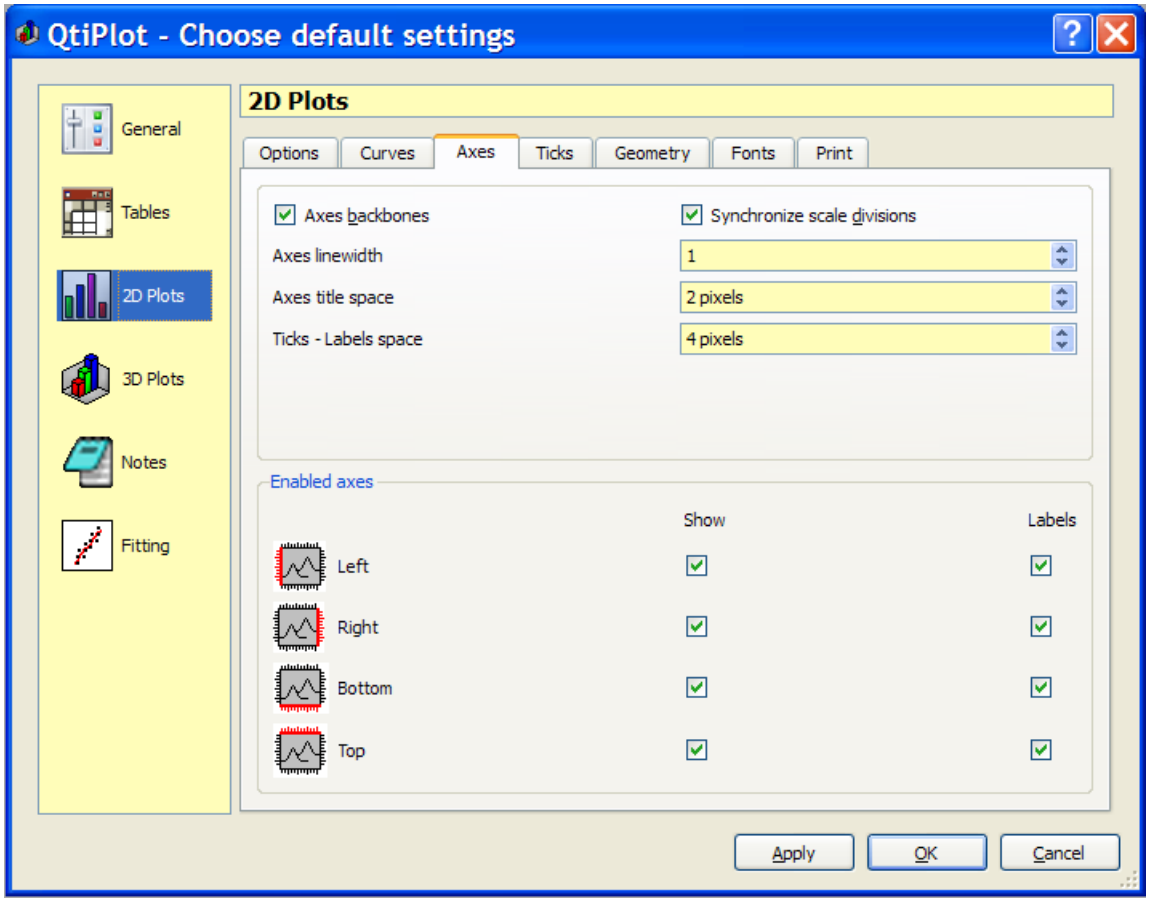


Figure 5.54: The preferences dialog: 2D plot options.

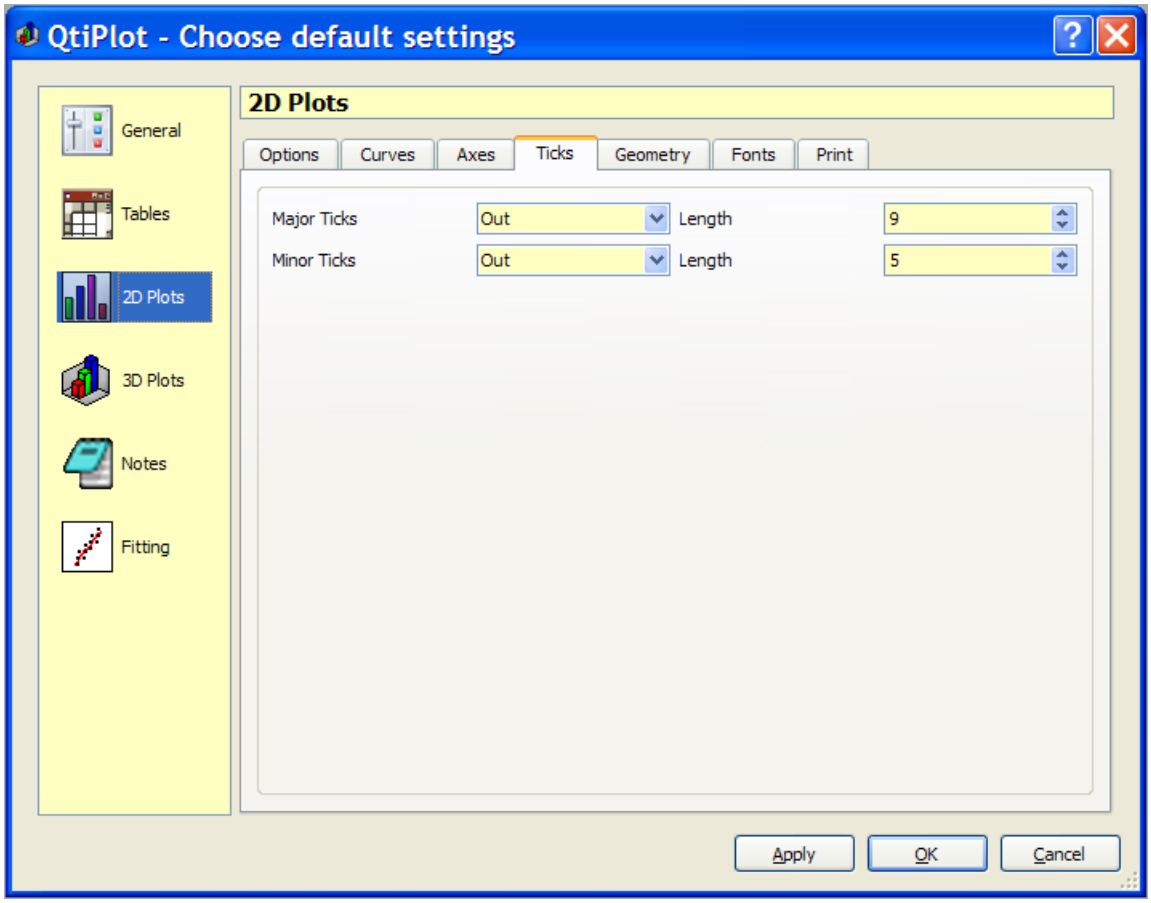
The second tab named *Curves* defines the default style used when you create a new plot.



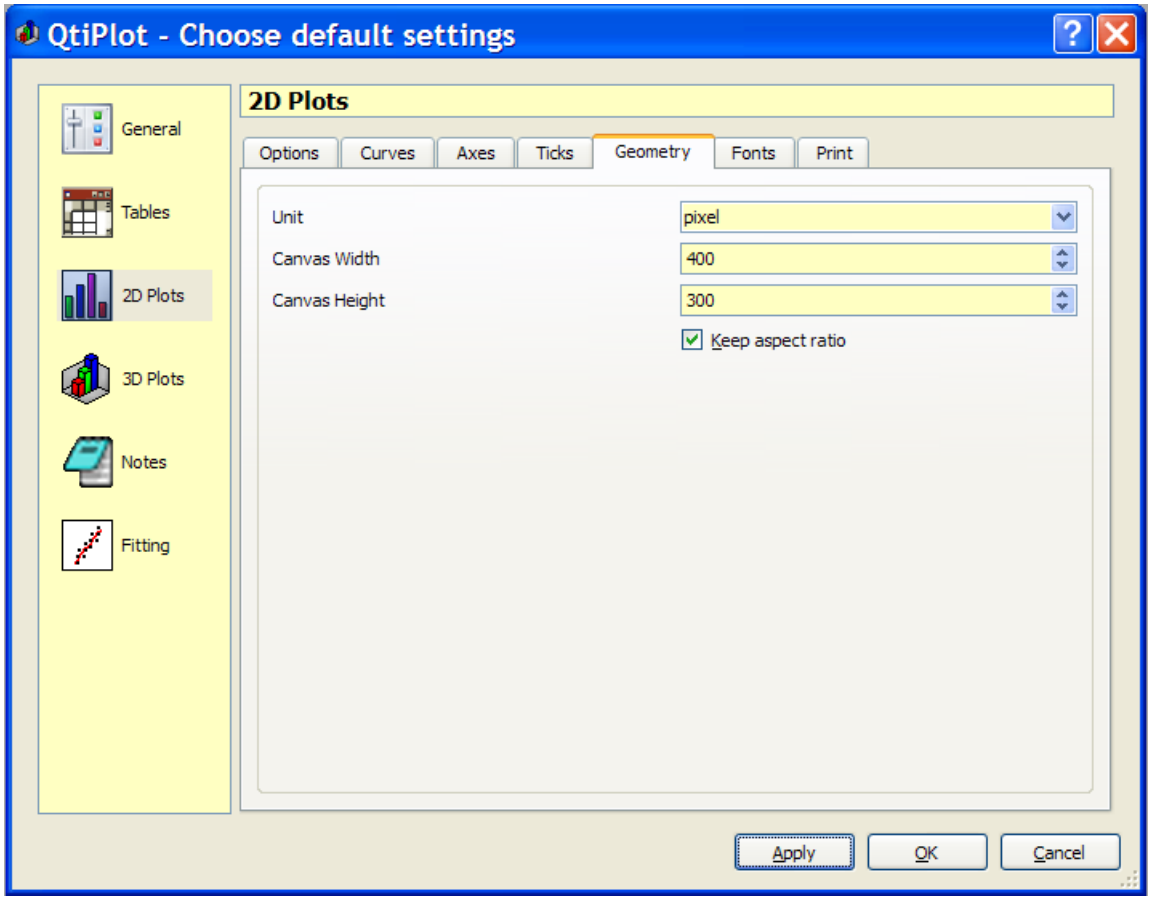
The third tab named *Axes* allows you to specify which plot axes will be displayed in a new layer and the main aspect parameters of an axis.



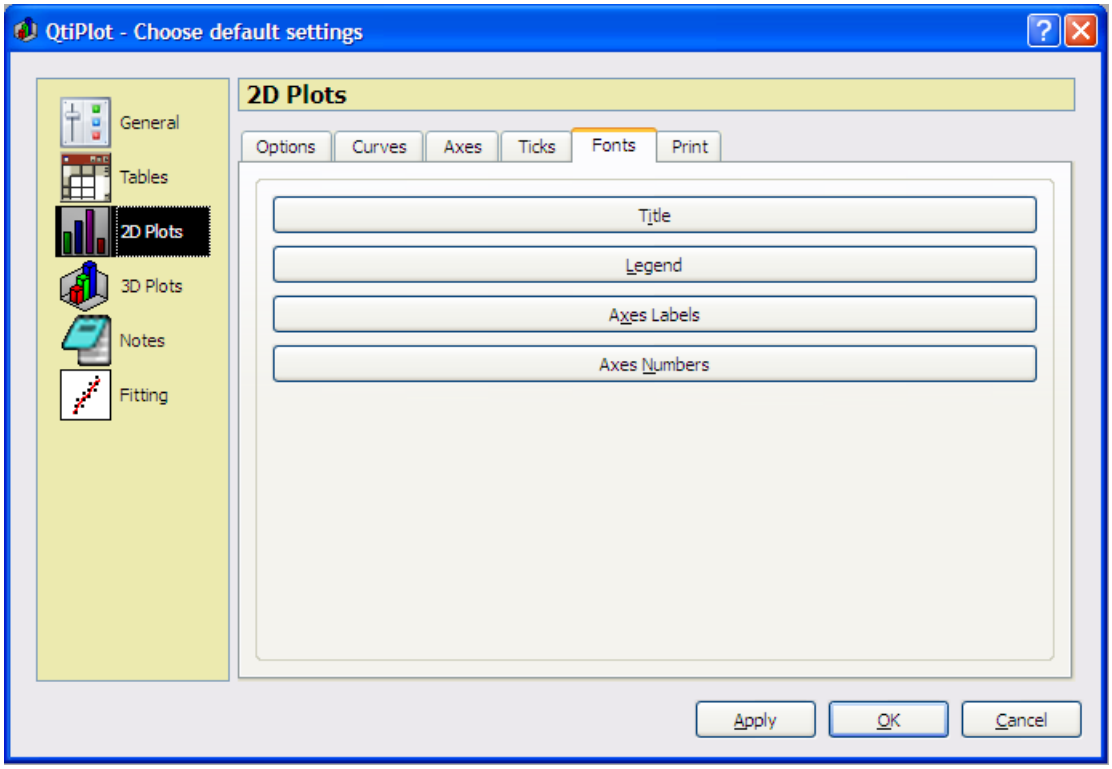
The fourth tab named *Ticks* defines the default style for the ticks of the axes used when you create a new plot.



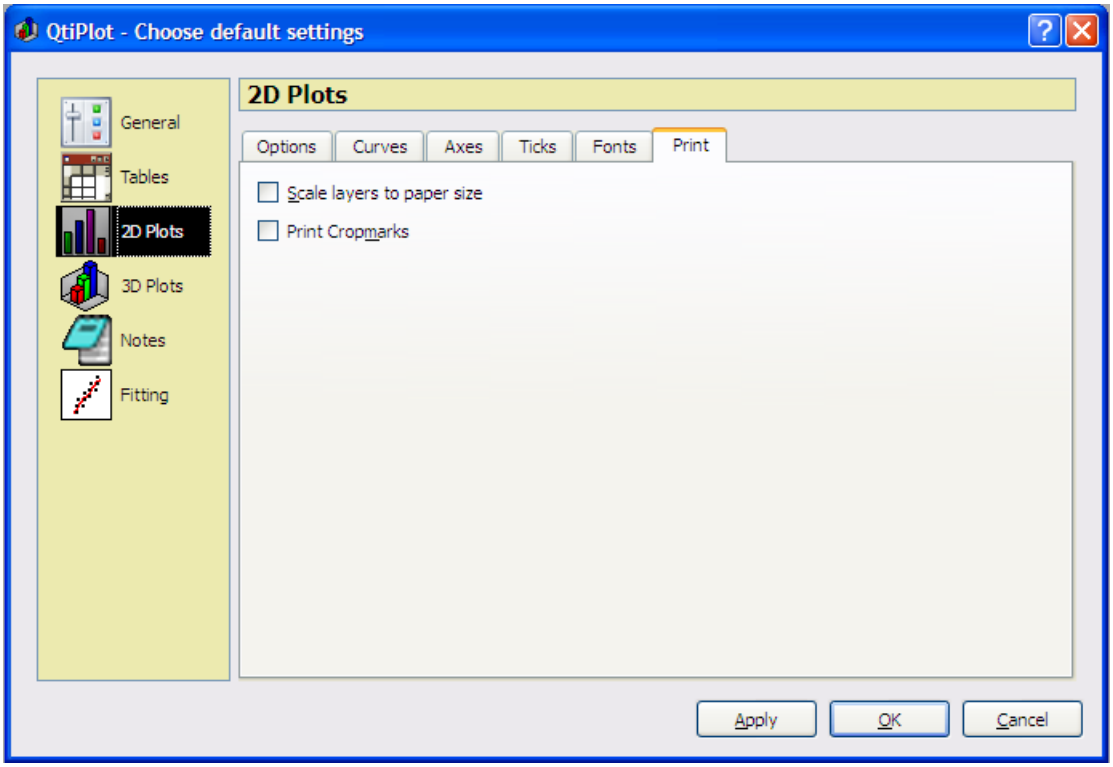
The fifth tab named *Geometry* defines the default size for the drawing area of a plot layer.



The sixth tab named *Fonts* defines the default fonts used when you create a new plot.



The last tab named *Print* allows you to define the default options used when printing 2D plots. If you want the layers to be printed with their original dimensions, you must be sure to uncheck the option *Scale layers to paper size*. By checking the *Print Cropmarks* option you ensure that some visible marks are drawn around the borders of the plot.



The following tab allows to customize the aspect of three dimensional plots. From this dialog you have the possibility to define a speed drawing mode, which is very usefull when working with large data sets. This can be realized via the *Resolution* option: the higher this value, the smaller the number of data points drawn on the 3D plots, therefore the higher the drawing speed. For a value of 1, all the data points are drawn.

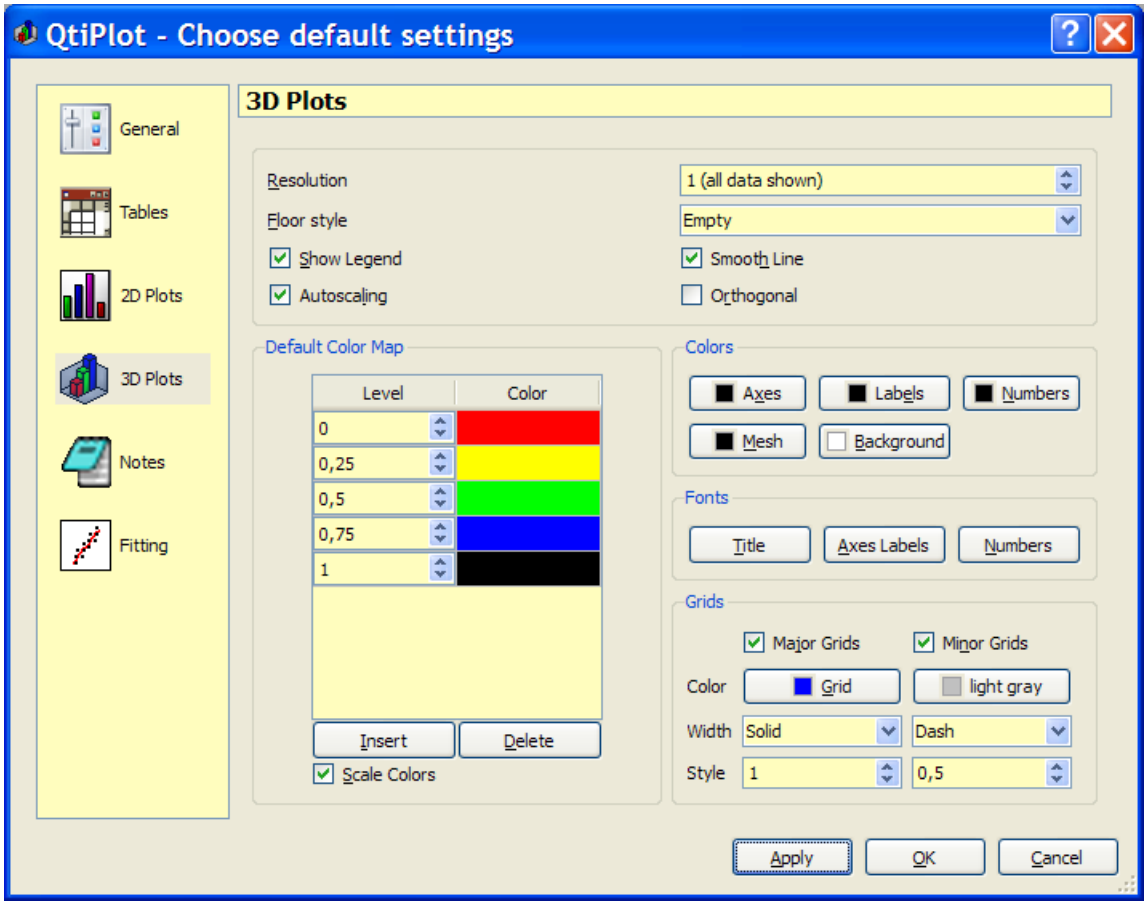


Figure 5.55: The preferences dialog: 3D plot options.

The *Notes* tab allows the user to customize some default options for the text editors, such as the length of the *TAB* character and the font. The user can also specify whether the line numbers should be displayed or not. Displaying the line numbers can be particularly helpful when debugging Python scripts.

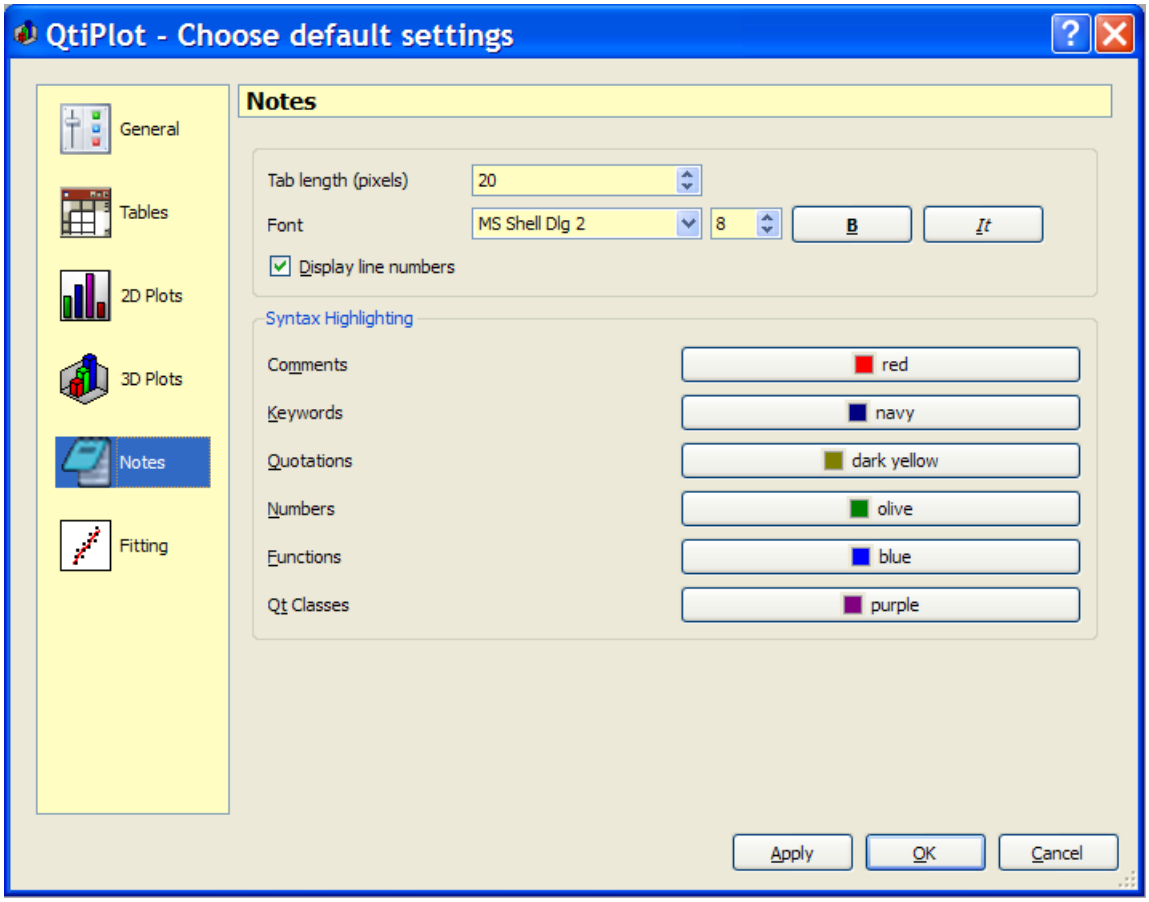


Figure 5.56: The preferences dialog: note options.

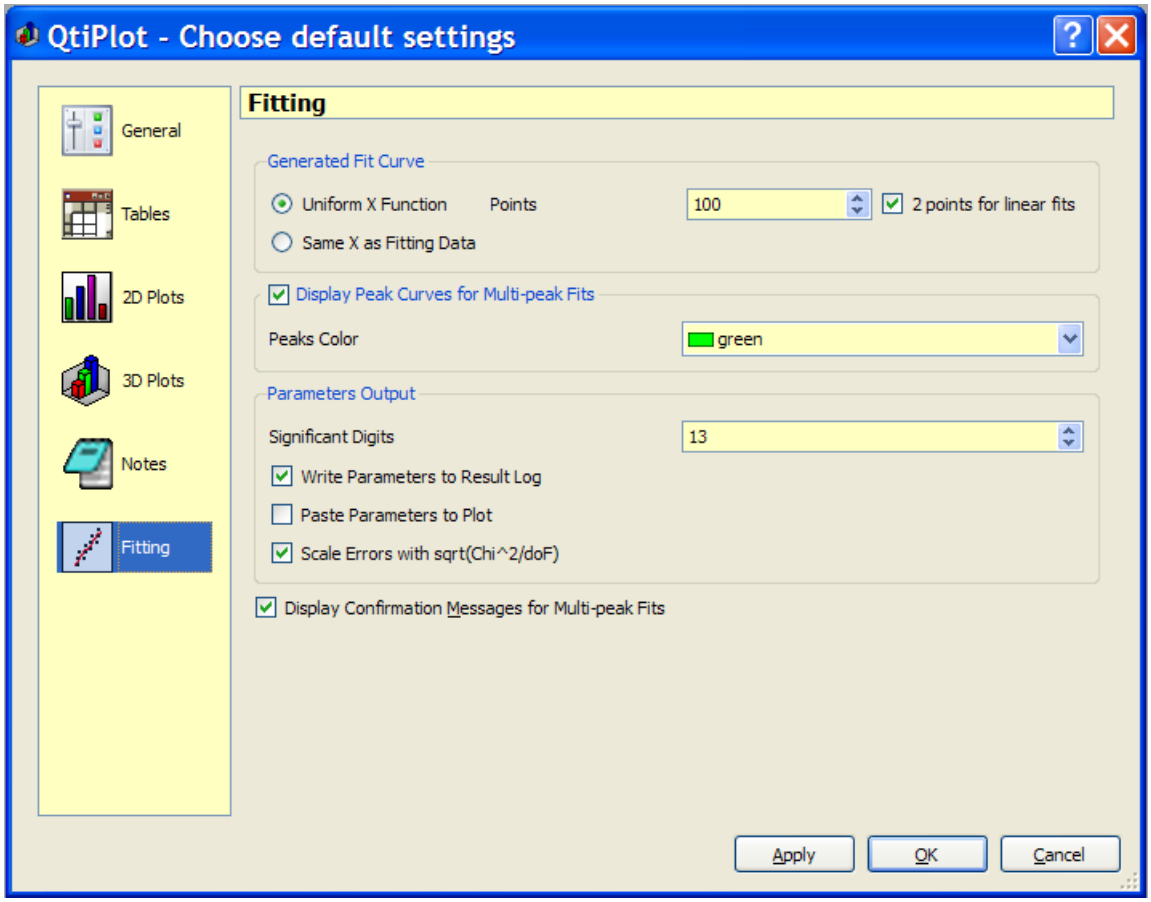


Figure 5.57: The preferences dialog: fitting options.

5.20 Printer-setup

This dialog box is opened by the Print command from the File menu. It is used to print the selected window (plot or table) and its aspect depends on your operating system. The following screenshot shows this dialog on a Linux system using the KDE window manager.

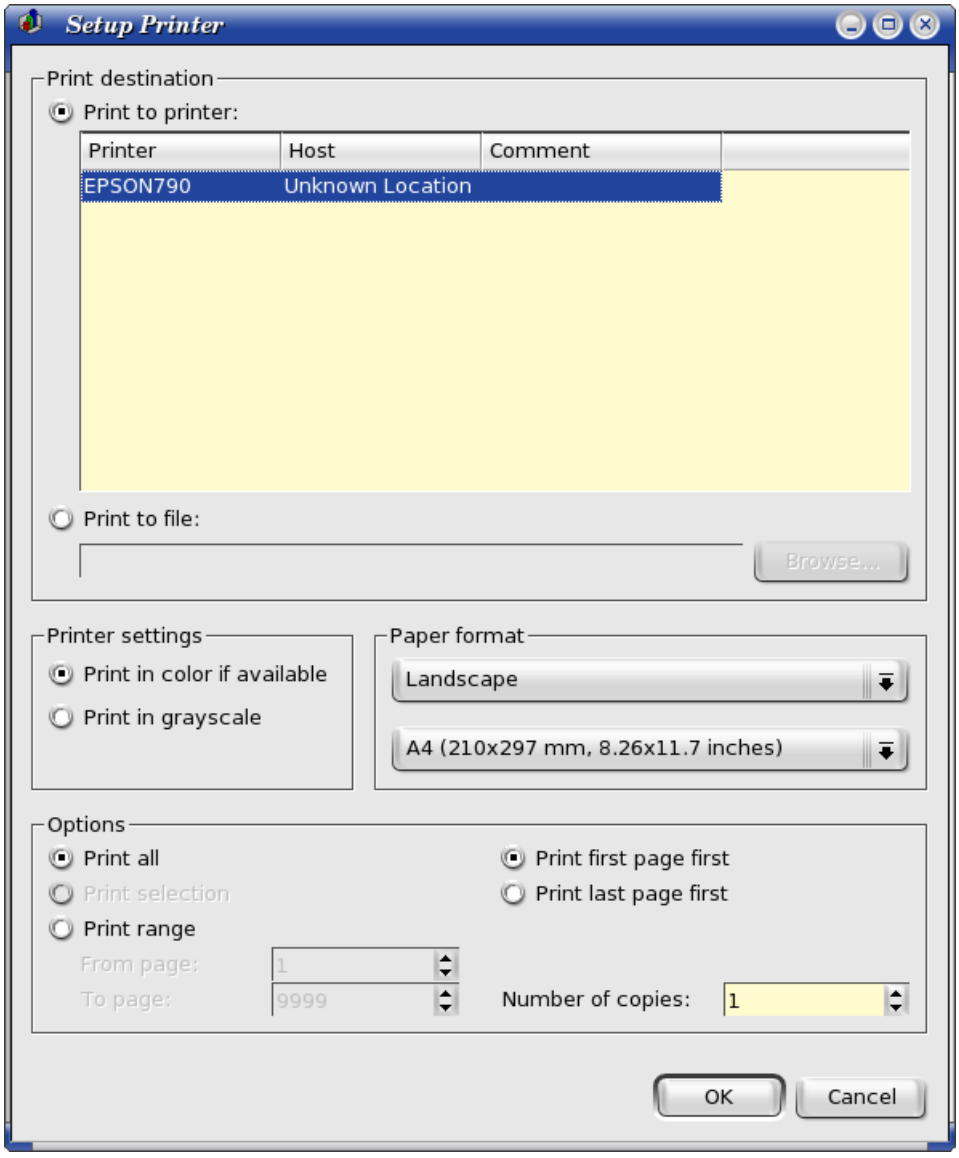


Figure 5.58: The **Print** dialog.

5.21 Set Column Values

This dialog is activated by the Set Column Values... command of the Table menu. It allows to fill a column with the result of a function.

The available mathematical functions (assuming you are using the default script-

ing language, muParser) are listed in the appendix. 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 here. Nevertheless, even if Python allows for a more powerfull synthax, it can be quite slow for very large tables. Therefore you have the possibility to use muParser instead, even if Python is set as the default script engine for your current project, by checking the box *Use built-in muParser*. Using muParser as scripting engine higly increases the speed of the evaluation for single line expressions.

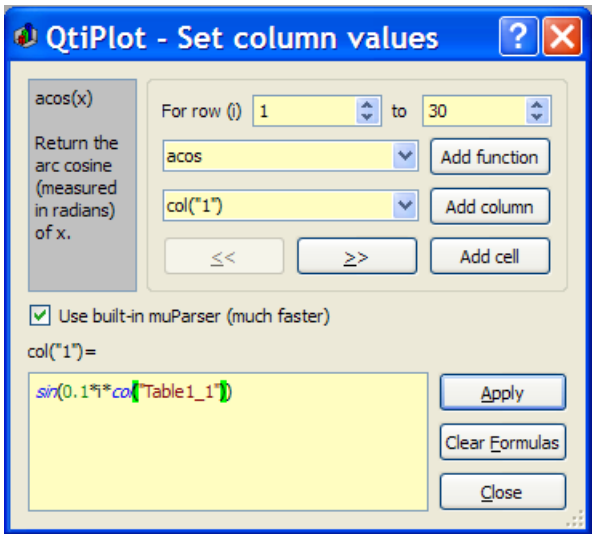


Figure 5.59: The **Set Column Values...** dialog.

Warning: If you make some changes in the table, the values are not computed again, unless you check the *Automatically Recalculate Column Values* option in the *Tables* tab of the Preferences dialog. If this option is unchecked you have to explicitly tell QtiPlot to recalculate individual cells or whole columns or rows by selecting "Recalculate" from their context menu or by pressing Control+Return.

5.22 Set Matrix Dimensions

This command is in the Matrix menu. It allows to specify the number of rows and columns of a matrix. In this window, you can also define a range for X-values and Y-values. These X and Y ranges will be used in plots as initial scale ranges. They are also available via the *x* and *y* variables if you choose to define the content of the matrix with the Set Values Dialog.

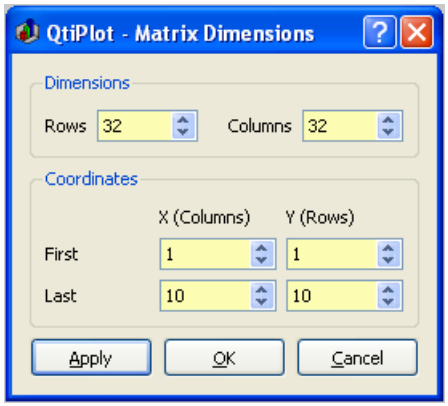


Figure 5.60: The **Set Dimensions...** dialog for matrix.

5.23 Import ASCII files

This dialog is activated by selecting the command Import -> Import ASCII... from the File Menu. It can be used to select several files at a time. The files are imported using a set of default options. You can display and edit these options by pressing the *Advanced* button. All your changes to the default import parameters will be saved.

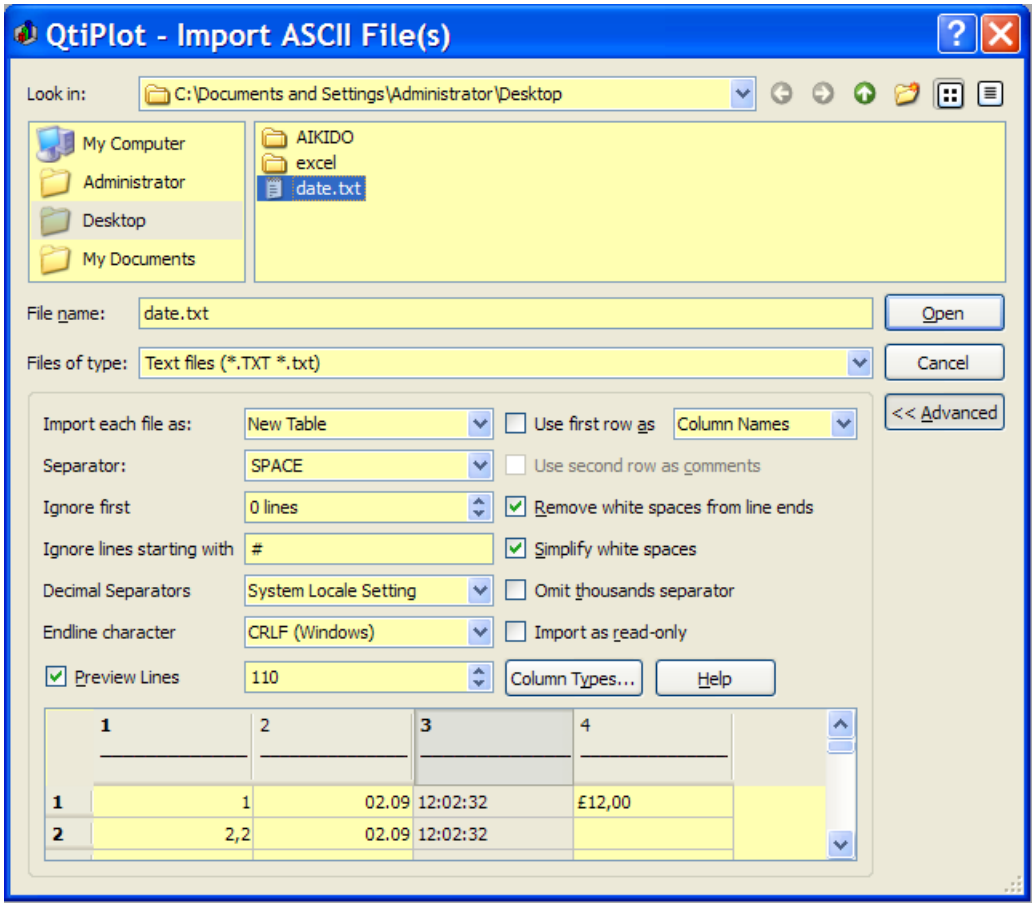


Figure 5.61: The dialog box.

You can define a custom file filter in the *Files of type:* field in order to preselect files (e.g. typing *.mydata then "Enter" so that only files XXX.mydata are listed in the directory view).

Concerning the separator which is used between the columns, you must bear in mind that there is no grouping of separators, so if you use "SPACE", you must use only *one* separator between each column or check the *Simplify white spaces* option.

If your data files have a particular structure, for example if they present several lines containing the description of your experiment, before the actual data start, you have the possibility to skip the n first lines of the file. You can also skip all lines starting with a comment character/string. The comment string can be customized via the text edit line *Ignore lines starting with*.

If you choose to use the first line as column names and/or the second line as comments to be displayed in the table header, you must use the same separator between the

column names and between the data columns.

Assuming that the data files you want to import were created using different decimal separator conventions than the ones you're using, like for example a comma character for the decimal point instead of a dot character, you have the possibility to convert the data to your current settings, by specifying the *Decimal Separators* convention used in the ASCII files in conjunction with the *Omit thousands separator* option.

Finally, you can specify the kind of data to be imported in each column by pressing the *Column Types...* button or by clicking on the column header in the preview table. A dialog will pop-up allowing you to choose a data type (the default data type for each column is *Numeric*) and a column format for date/time columns.

5.24 Matrix Properties

This command is in the Matrix menu. It allows to specify some global properties of the selected matrix such as the cell width (in pixels) and the format for numbers.

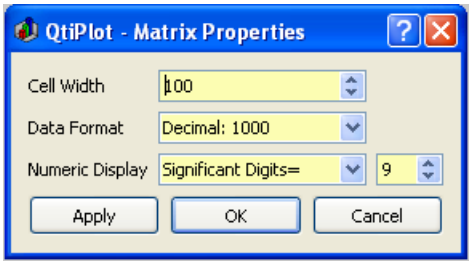


Figure 5.62: The **Set Properties...** dialog for matrix.

5.25 Set Matrix Values

This command is in the Matrix menu. It allows to 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.

You can use the X-values and Y-values defined with the Set Dimensions... command command, and also define your functions based on the *x* and *y* variables.

The functions can be written on several lines, and the intrinsic functions which are available are listed in the appendix .

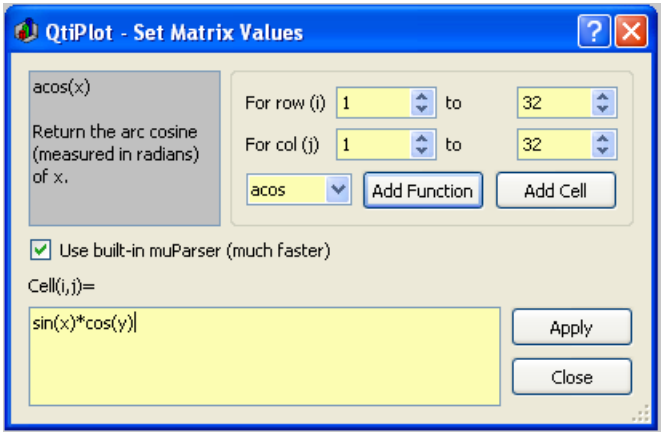


Figure 5.63: The **Set Values...** dialog for matrix.

Using Python as scripting engine for the calculation of matrix values has the drawback that even if it allows a more powerfull synthax, it can be quite slow for large matrices. Therefore you have the possibility to use muParser instead, even if Python is set as the default script engine for your QtiPlot project, by checking the box *Use built-in muParser*. Warning: muParser is very fast for the evaluation of single line expressions only, therefore try to avoid a synthax like:

```
a = cell(1, 1)
b = cell(2, 2)
a*b*x + b*x*x + a
```

and prefer the following one:

```
cell(1, 1)*cell(2, 2)*x + cell(2, 2)*x*x + cell(1, 1)
```

which will highly increase the speed of the evaluation!

5.26 Surface plot options

This dialog box is used to customize a 3D function plot which has been created by the New -> New Surface 3D Plot command from the File menu. It is activated by a double click on the 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.

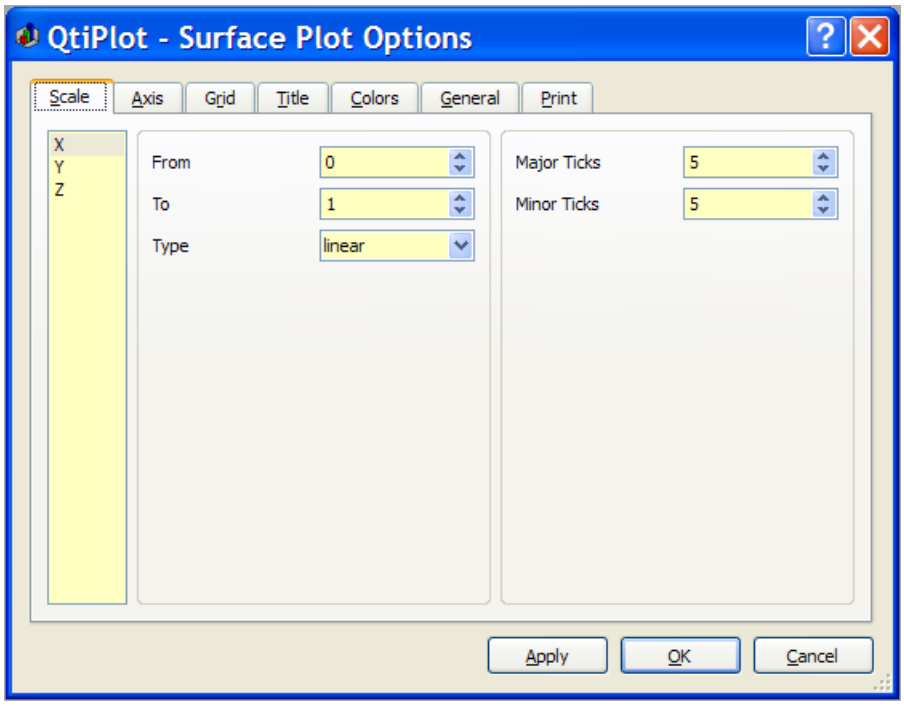
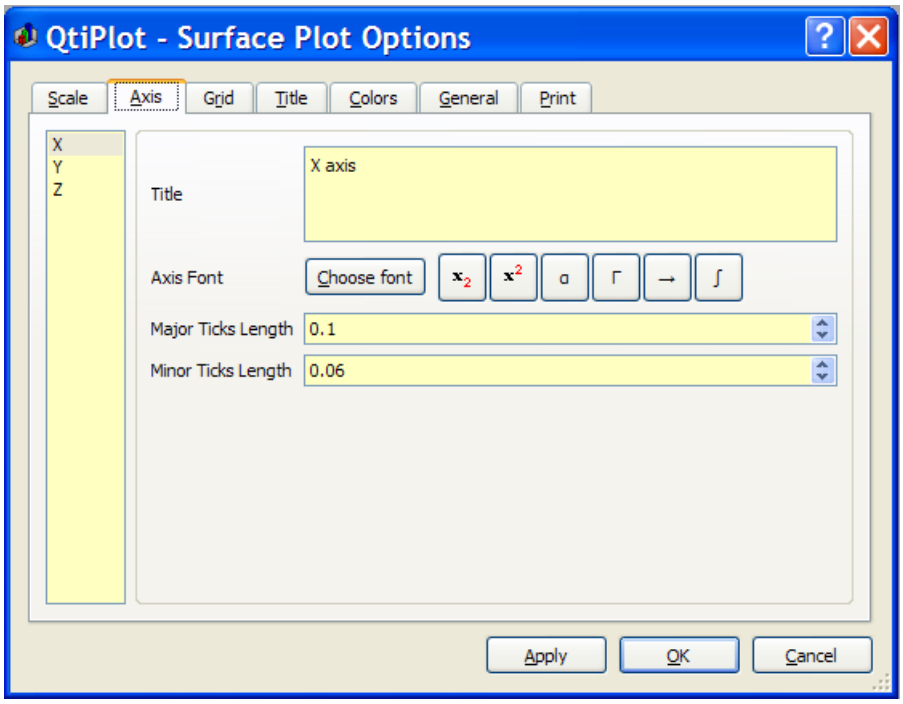
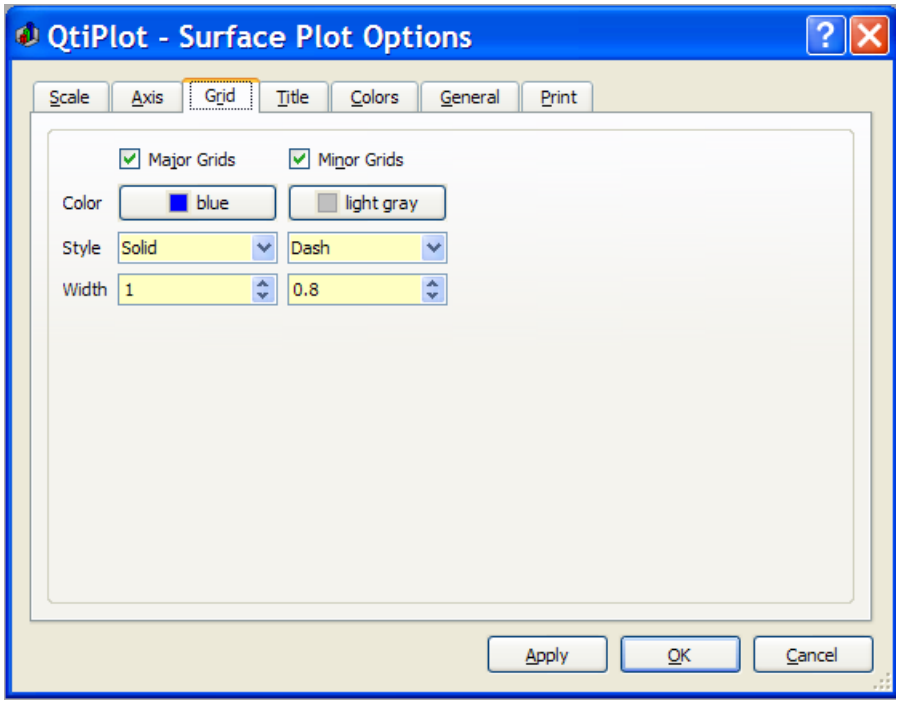


Figure 5.64: The surface plot options dialog box.

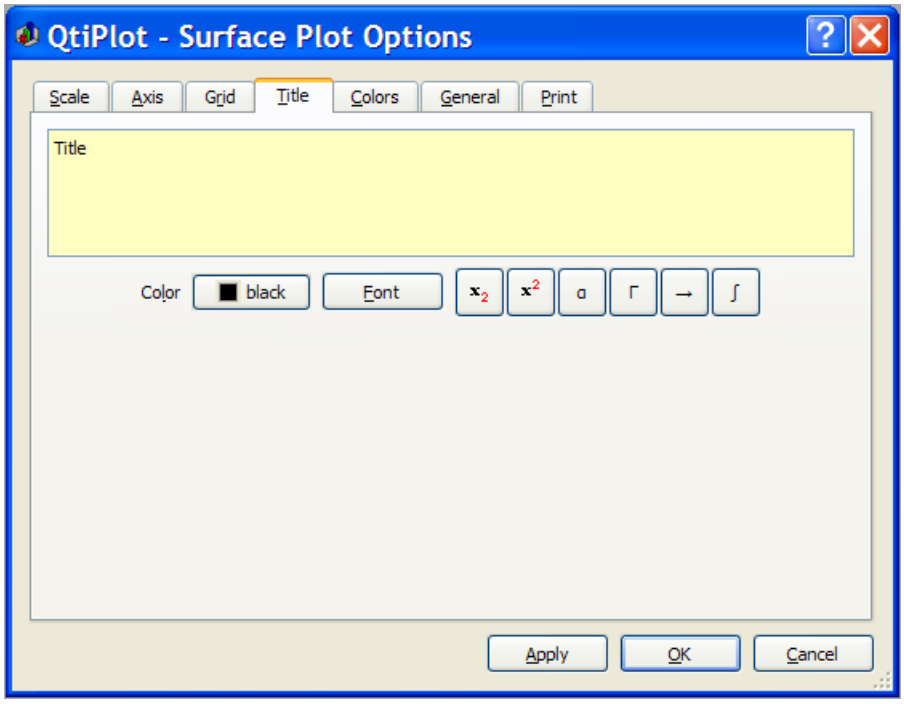
The second tab defines the main parameters of the three axis: the axis label and its font, and the length of the ticks. This length is defined in the same units as the range of the axis. If something is changed in the scales of the graph, the length of the ticks is re-calculated by QtiPlot. The font 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. The superscript and subscript buttons allow an easy insertion of LaTeX superscript/subscript commands. The superscript/subscript texts will only be rendered as such if you export the plot to a .tex file, by choosing the *LaTeX file* text mode in the advanced settings of the export dialog. In order to render the superscripts/subscripts, this file must be compiled separatelly, therefore you need a LaTeX environement installed on your computer.



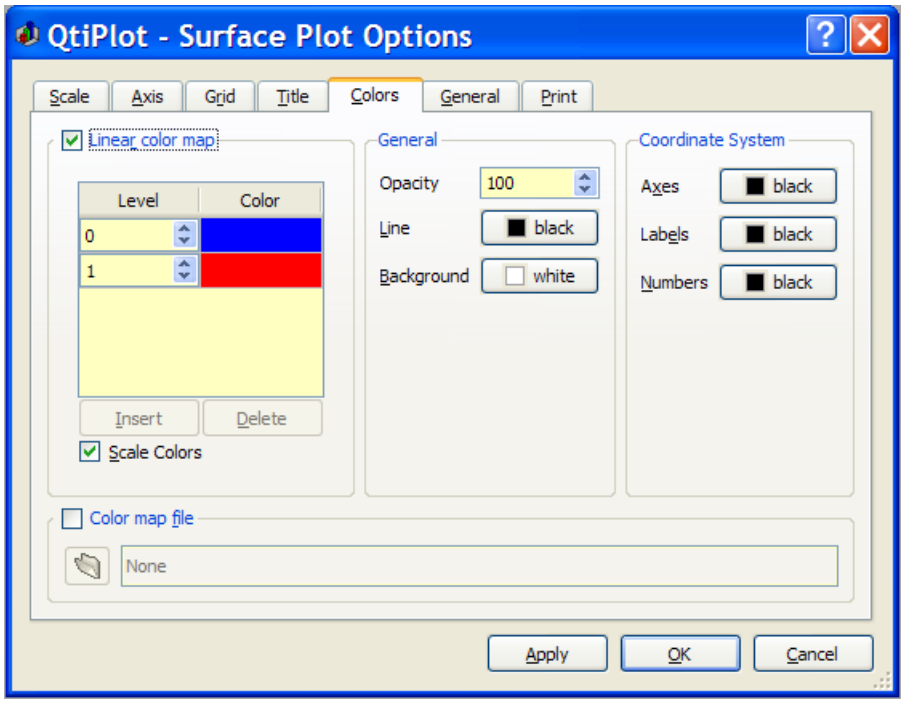
The third tab is used to define or modify the properties of the plot grid.



The fourth tab is used to define or modify the title of the plot. You can not add subscripts/superscripts, bold characters, etc in your title as you can do it for 2D plots.



The fifth tab allows to modify the colors used in the different elements of the plot.



The *Linear color map* group defines the color scheme which is used to show the Z-values. If the *Scale Colors* box is checked, a Z value will be represented by a color defined as a linear interpolation between the adjacent values in the color table.

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 QtiPlot web site.

The *General* tab is used to define some global parameters and the aspect ratio of the plot. 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.

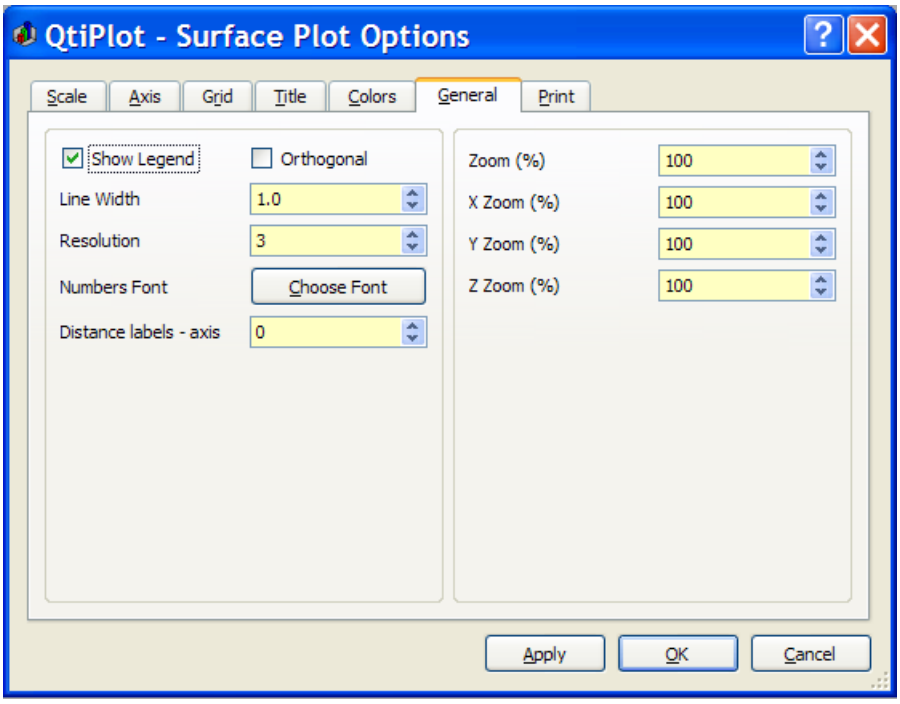


Figure 5.65: The general plot options tab.

The *Print* tab is used to define some useful print options, like scaling to the paper size or showing the cropmarks.

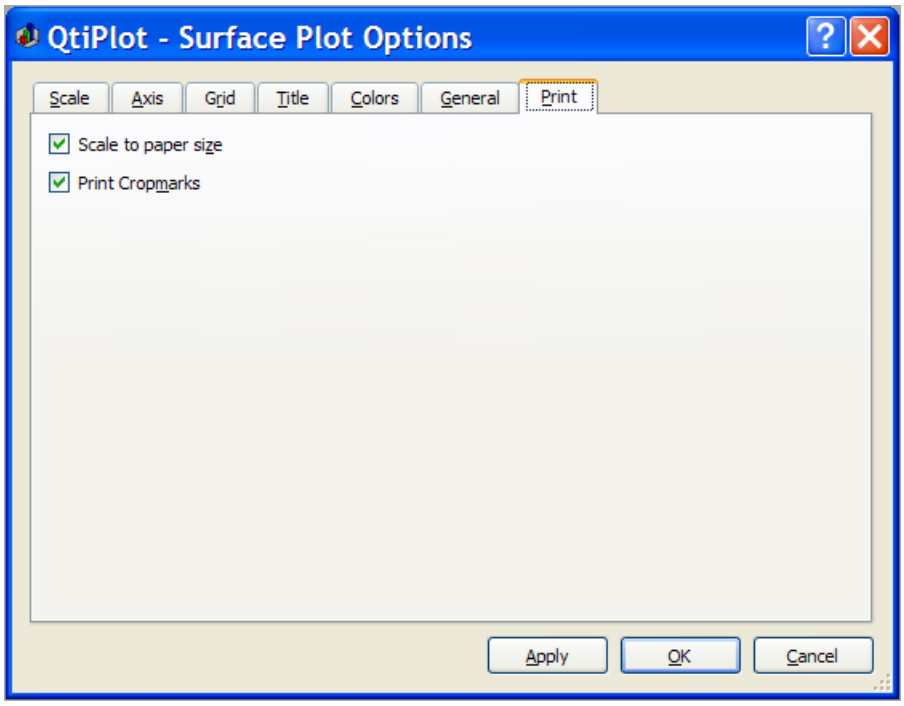


Figure 5.66: The 3D plot print options.

5.27 Text options

This dialog can be opened by several commands such as Title... command or when you double click on an axis title in your plot. The *Color*, *Font* and *Alignment* commands allow the modification of the general settings of the text label. You can also specify the distance between the label and the corresponding axis, using the *Distance to axis* box.

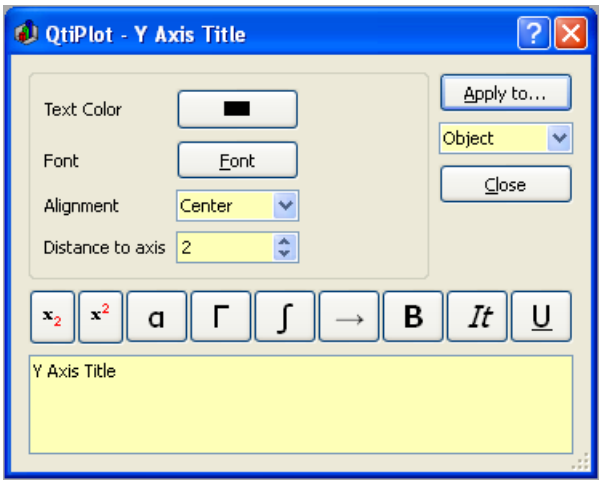


Figure 5.67: The axis title options dialog.

The following slightly modified dialog is opened when you double click on texts/legends in your plot in order to customize them. It also allows to add text boxes to a plot layer. When the text object is a legend, listing the curves plotted on the layer, the curve symbols and lines are also drawn in the text box, in front of the curve names. In order to display the symbol of a curve and the name of the data set, the follwing synthax is used: *\l(1.2)%(1.2)*. In this example the first integer before the dot is the index of the plot layer and the second value, the number 2, refers to the index of the curve. The index of the layer is optional, if not specified, the parent layer is used.

The % character is an alias for the name of the data set. By adding ",@C" or ",@L" to the number you can alternativly use the name or comment of the dataset, e.g. *\l(2)%(2,@C)* will use the column name. Additionaly you can use "@W" to display the table name and "@WL" for the table label. To display the contents of a specific cell of the source data table use *%(curve #,@L,col,row)*. If the col parameter is missing the y-column of the dataset is used.

If the *Auto-update* box is checked the legend is updated each time a curve is added or removed from the plot layer.

The *TeX Output* option specifies if LaTeX special characters should be escaped or not when exporting to .tex files. If the text contains LaTeX syntax (like superscripts, subscripts, etc...) and you want them to be interpreted by the LaTeX compiler, you should check this option.

By pressing the *Set As Default* button all text format options will be saved to the user preferences and will be applied to new text objects. The *Apply to...* button can be used to apply the format options to all text objects in the active plot layer or in the current plot window or in all plot windows in the project. The domain of application can be chosen from the list box bellow the *Apply to...* button.

The *Opacity* of the *Background color* can have a value between 0 (transparent background) and 255 (completely opaque background).

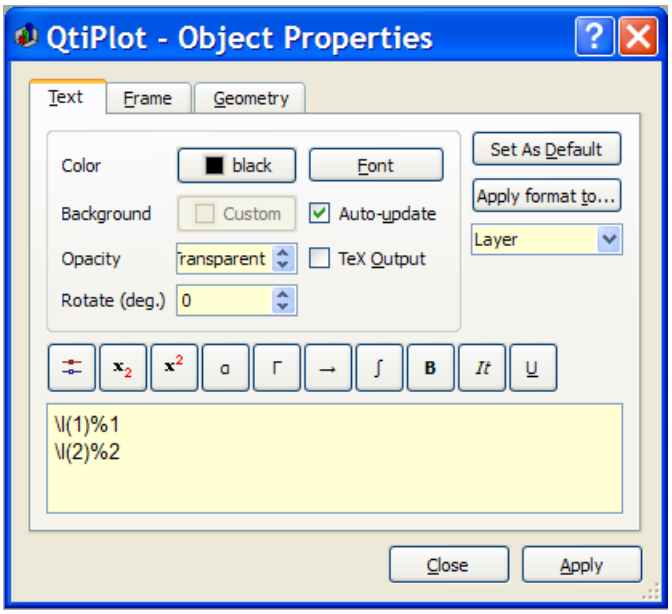
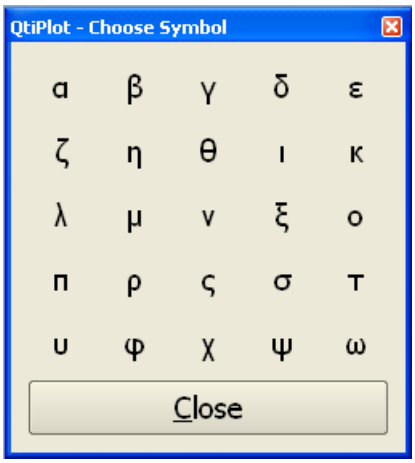


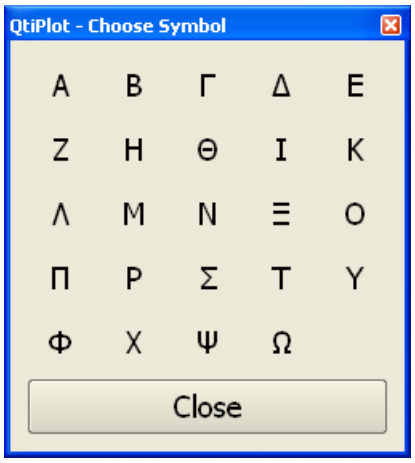
Figure 5.68: The legend/text options dialog.

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

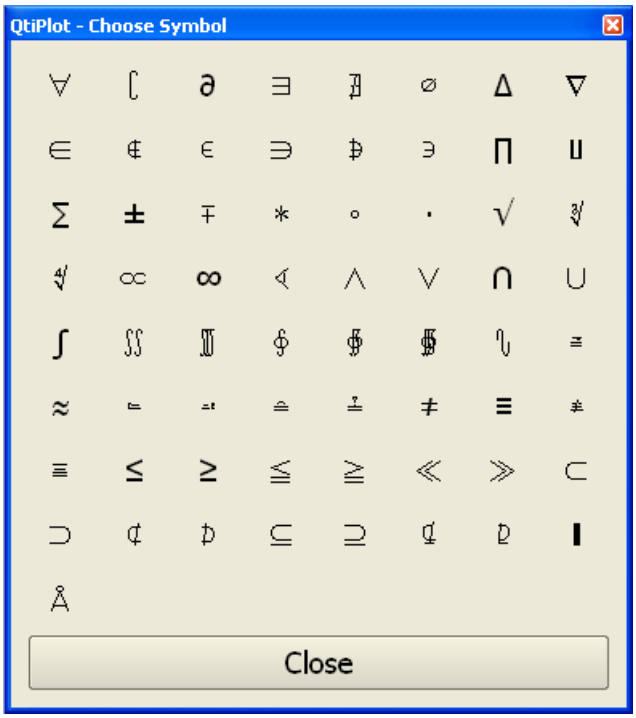
- _{text} will draw the text as subscripts. You can insert this sequence by clicking on the .
- ^{text} will draw the text as superscripts. You can insert this sequence by clicking on the .
- By clicking on the , you can open a new dialog which allows to select lower greek characters:



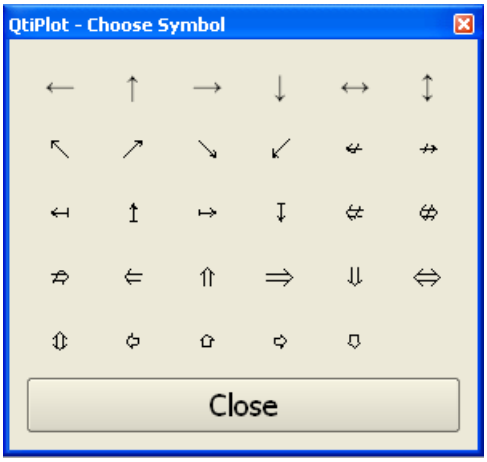
- By clicking on the Γ, you can open a new dialog which allows to select upper greek characters:



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



- By clicking on the arrow icon, you can open a new dialog which allows to select various arrow 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  .

Chapter 6

Analysis of data and curves

6.1 Fast Fourier Transform

This function can be accessed by the command FFT.... It can be found in the Analysis Menu when a table or 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 6.1.1: 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. QtiPlot will compute these parameters and build a new plot of the amplitude as a function of the frequency.

FFT performed on a curve to extract the characteristic frequencies. The signal is on the bottom plot, while the amplitude-frequency plot is on the top layer. In this example, the amplitude curve has been normalized, and the frequencies have been shifted to obtain a centered x-scale.

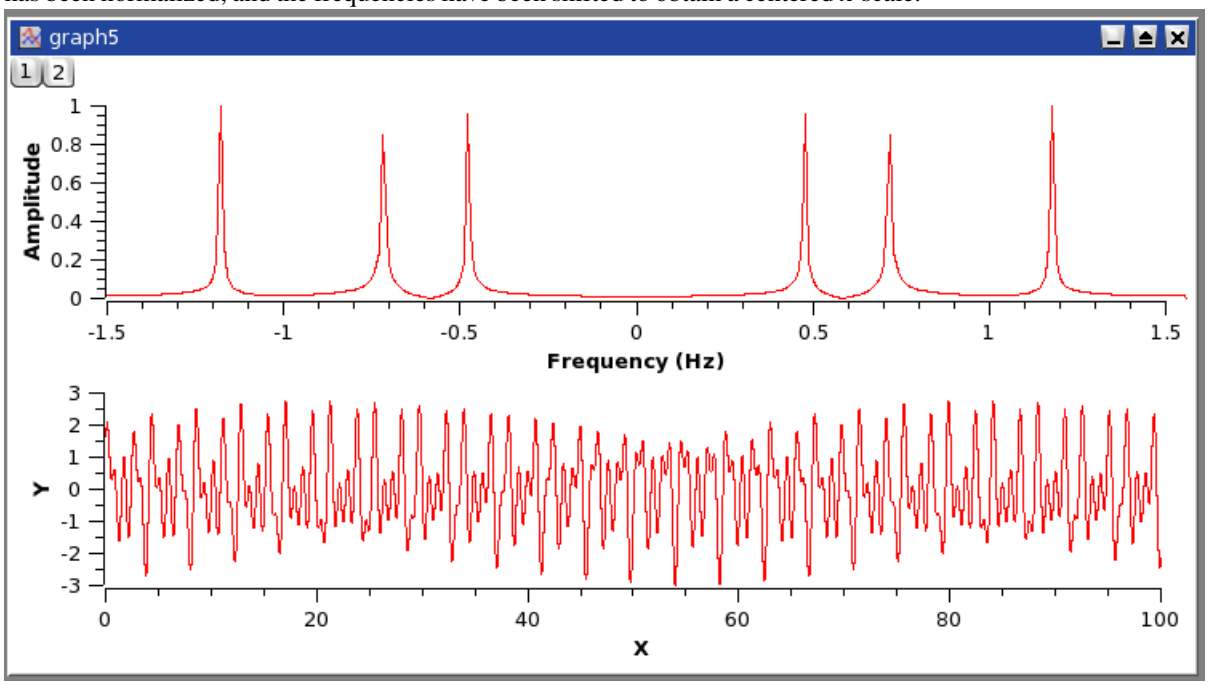


Figure 6.1: An example of a inverse FFT.

Some parameters of the FFT can be modified in the FFT dialog.

6.2 Correlation

This function can be accessed by the command Correlate. It can be found in the Analysis Menu when a table is selected. The correlation function, also known as the covariance function 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 6.2.1: Covariance function of two signals x(t) and y(t)

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 abscissae are therefore point numbers and not*t* values.

The first plot shows the two signals, the second one is the correlation function between the two signal which shows that there are correlations, and the third one is the Fourier transform which is done to extract the caracteristic frequencies of the correlation function.

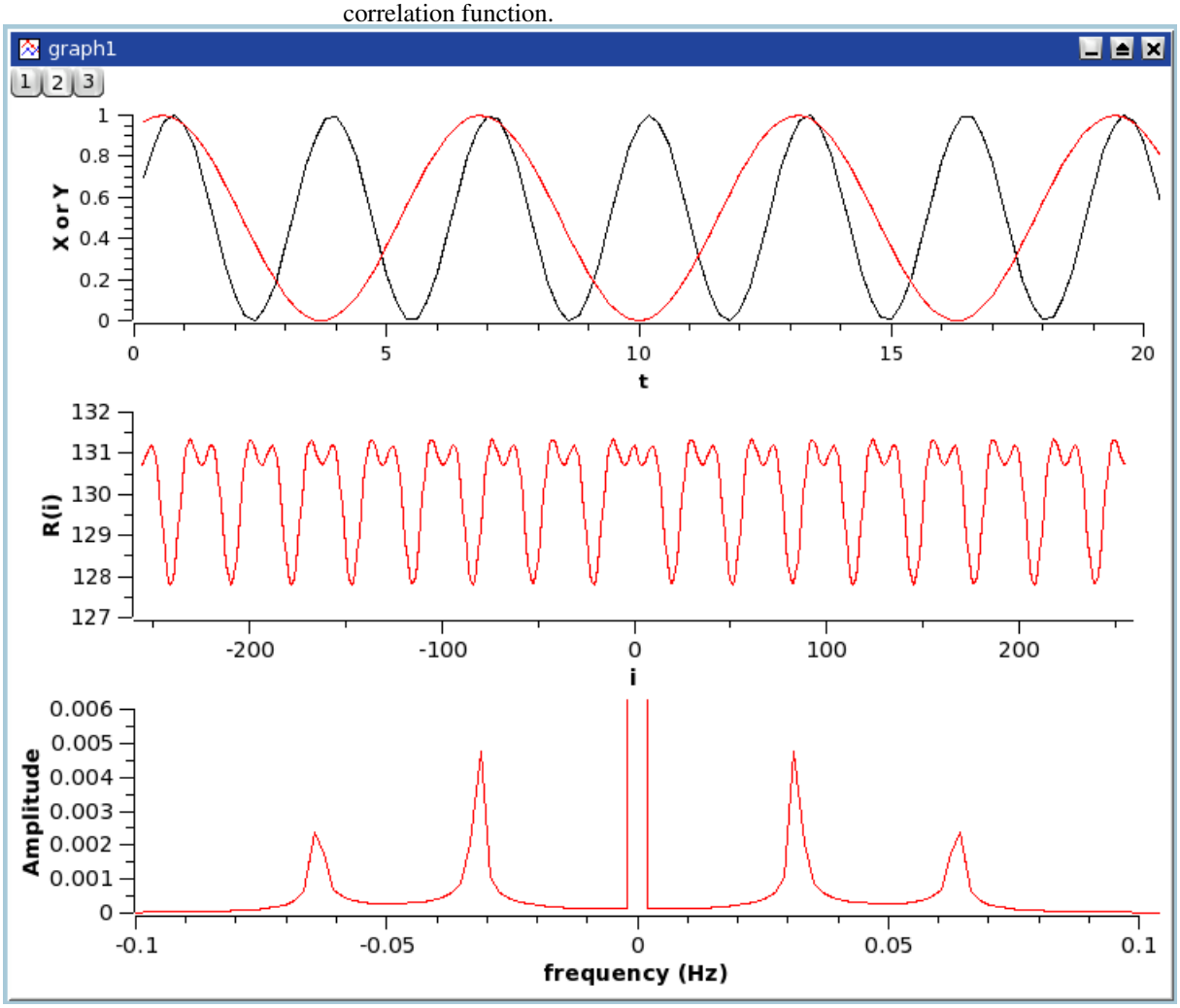


Figure 6.2: An example of a correlation between two sinus functions.

The correlation of a signal with itself can also be used in spectral analysis (it is then called autocorrelation or autocovariance function).

6.3 Convolution

.

6.4 Deconvolution

.

6.5 The Fit Wizard

This function can be accessed by the command Fit Wizard.... It can be found in the Analysis Menu when a table is selected.

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.

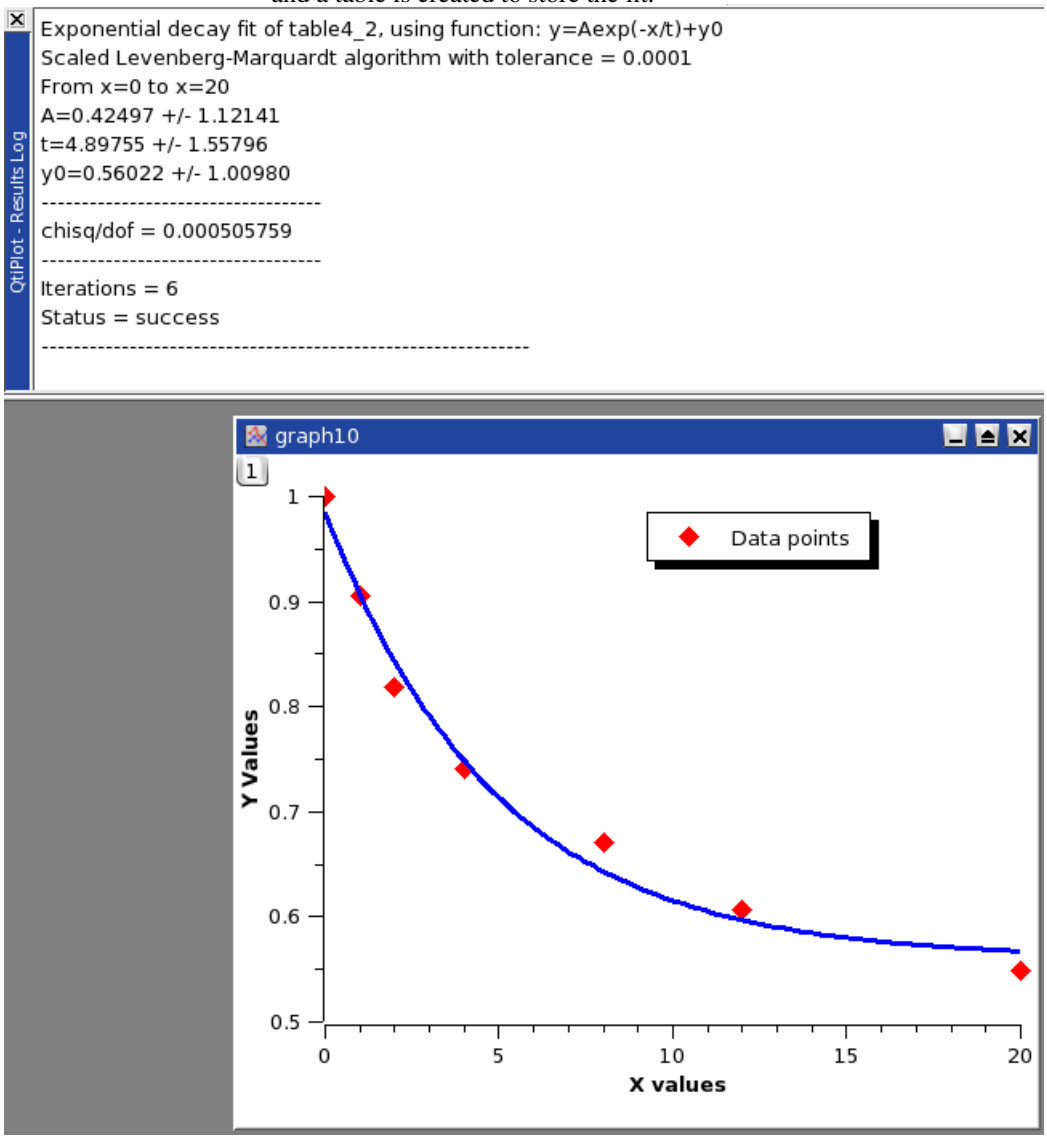


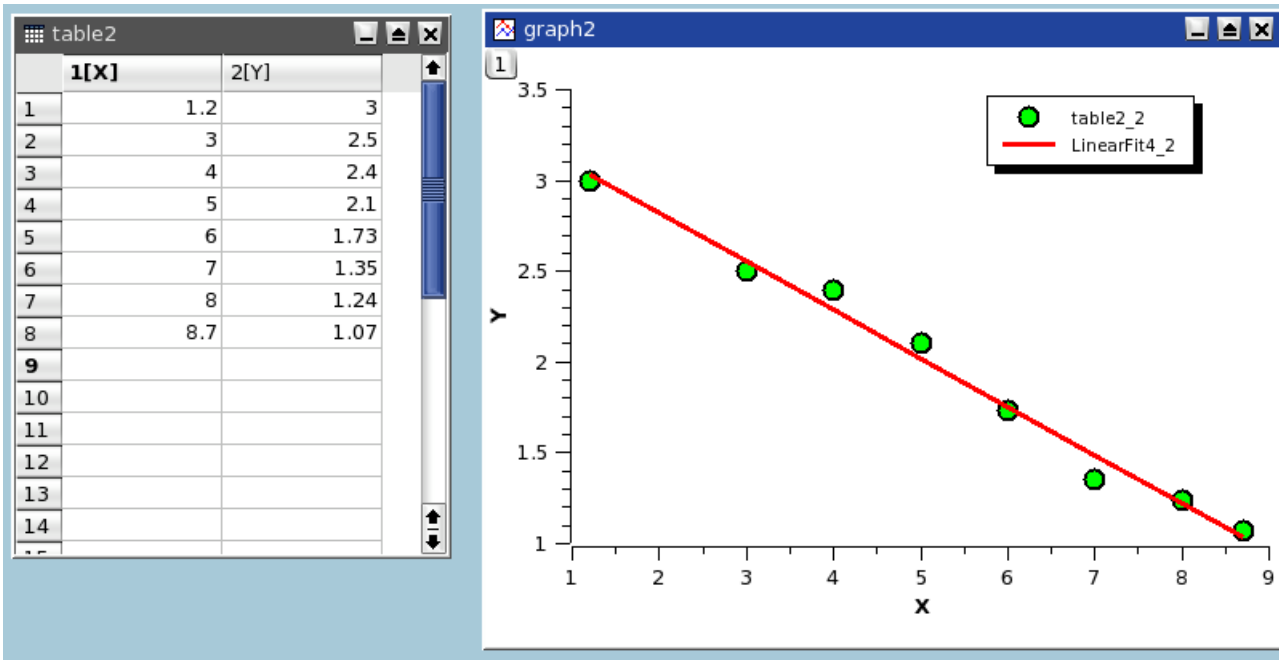
Figure 6.3: The results of the **Fit Wizard...**.

6.6 Fitting to specific curves

QtiPlot include quick access to the most usefull functions for fitting.

6.6.1 Fitting to a line

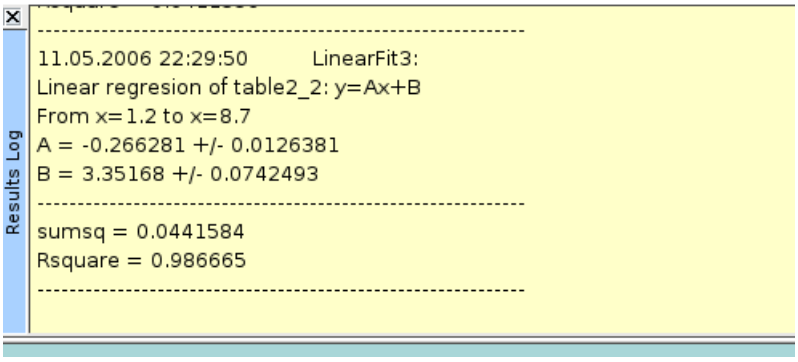
This command is used to fit a curve which has a linear shape.



	1[X]	2[Y]
1	1.2	3
2	3	2.5
3	4	2.4
4	5	2.1
5	6	1.73
6	7	1.35
7	8	1.24
8	8.7	1.07
9		
10		
11		
12		
13		
14		

Figure 6.4: The results of a **Fit Linear**.

The results will be given in the Log panel:



6.6.2 Fitting to a polynome

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

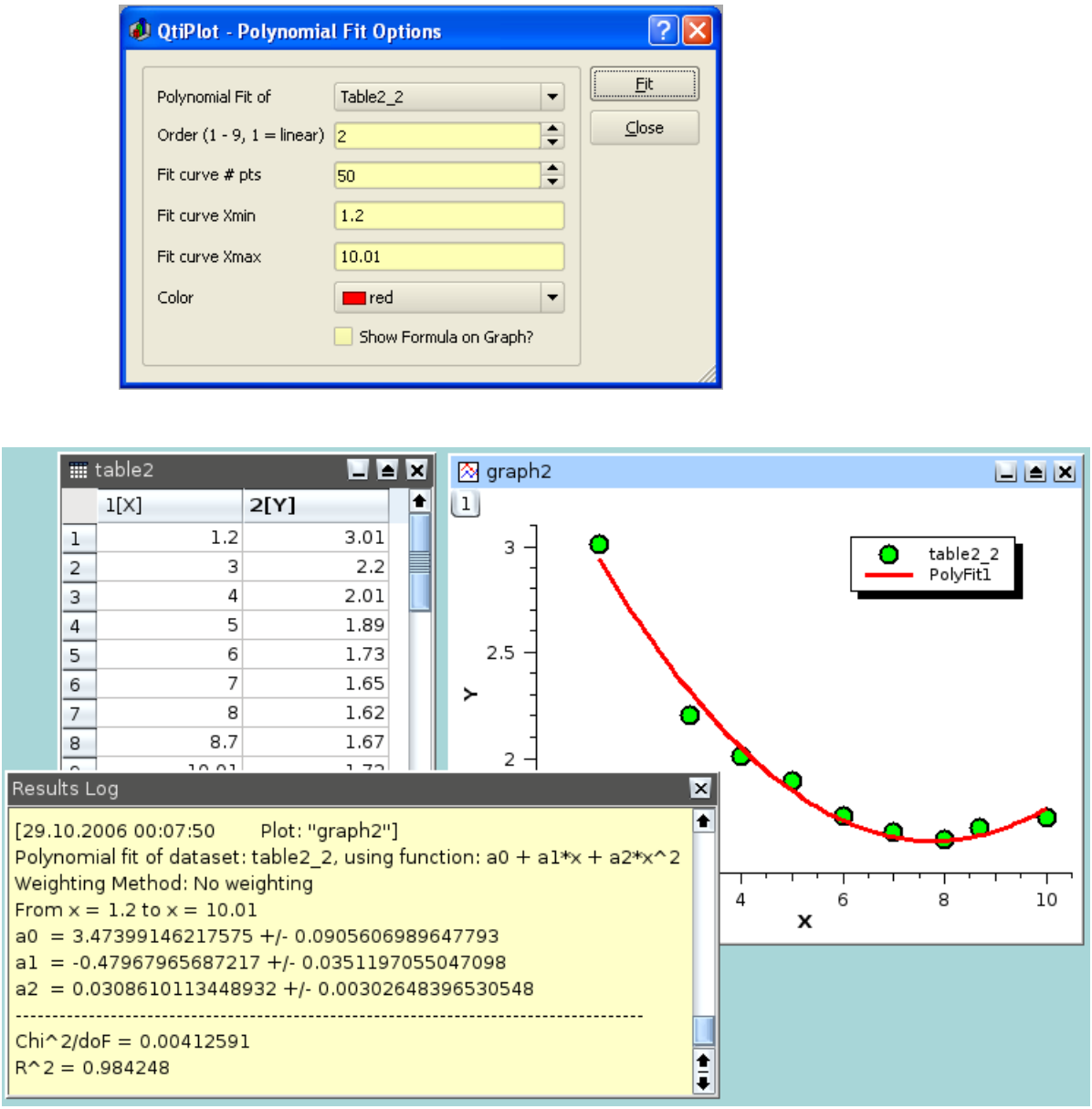


Figure 6.5: The results of a **Fit Polynomial...**, showing the initial data, the curve added to the plot, and the results in the log panel.

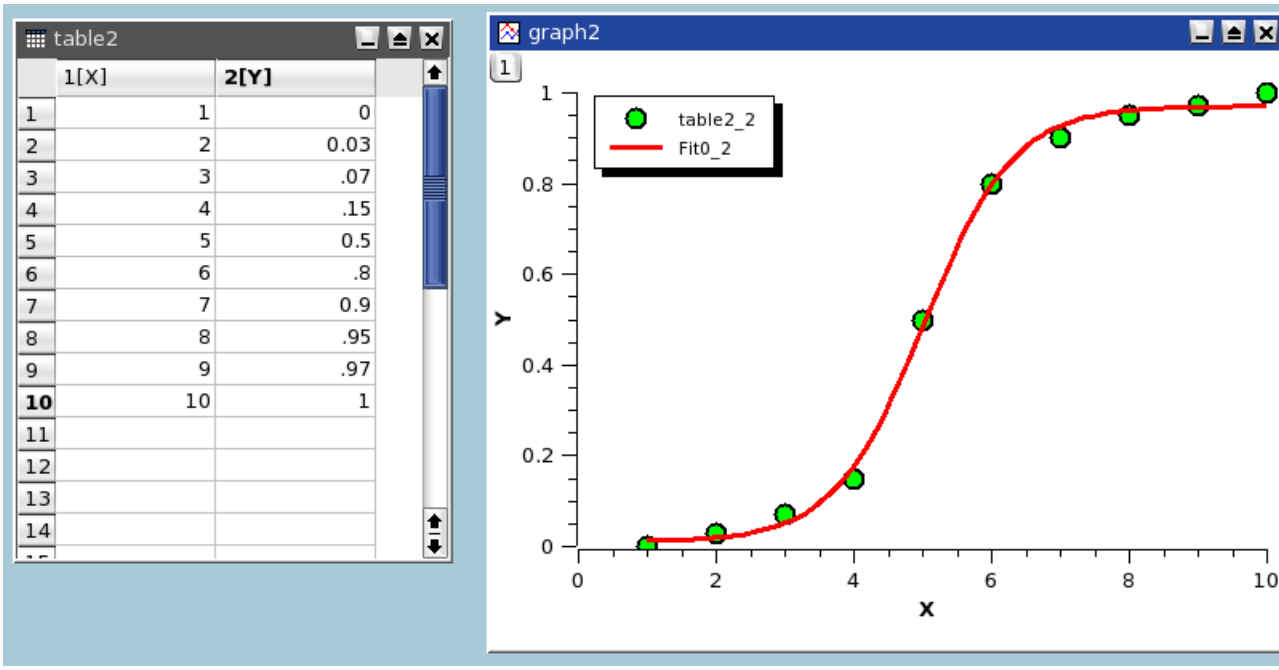
6.6.3 Fitting to a Bolzmann function

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

$$y = \frac{A_1 - A_2}{1 + e^{(x - x_0)/dx}} + A_2$$

EQUATION 6.6.1: Bolzmann equation

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



	1[X]	2[Y]
1	1	0
2	2	0.03
3	3	.07
4	4	.15
5	5	0.5
6	6	.8
7	7	0.9
8	8	.95
9	9	.97
10	10	1
11		
12		
13		
14		

Figure 6.6: The results of a **Fit Bolzmann (sigmoidal)**.

When the X axis is set to a logarithmic scale, the **Fit Bolzmann (sigmoidal)** command uses the Logistical equation for fitting:

$$y = \frac{A_1 - A_2}{1 + (x/x_0)^p} + A_2$$

EQUATION 6.6.2: Logistic dose response equation

where A_1 is the initial Y value, A_2 is the final Y value, x_0 is the inflexion point (center) and p is the power.

6.6.4 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 6.6.3: 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.

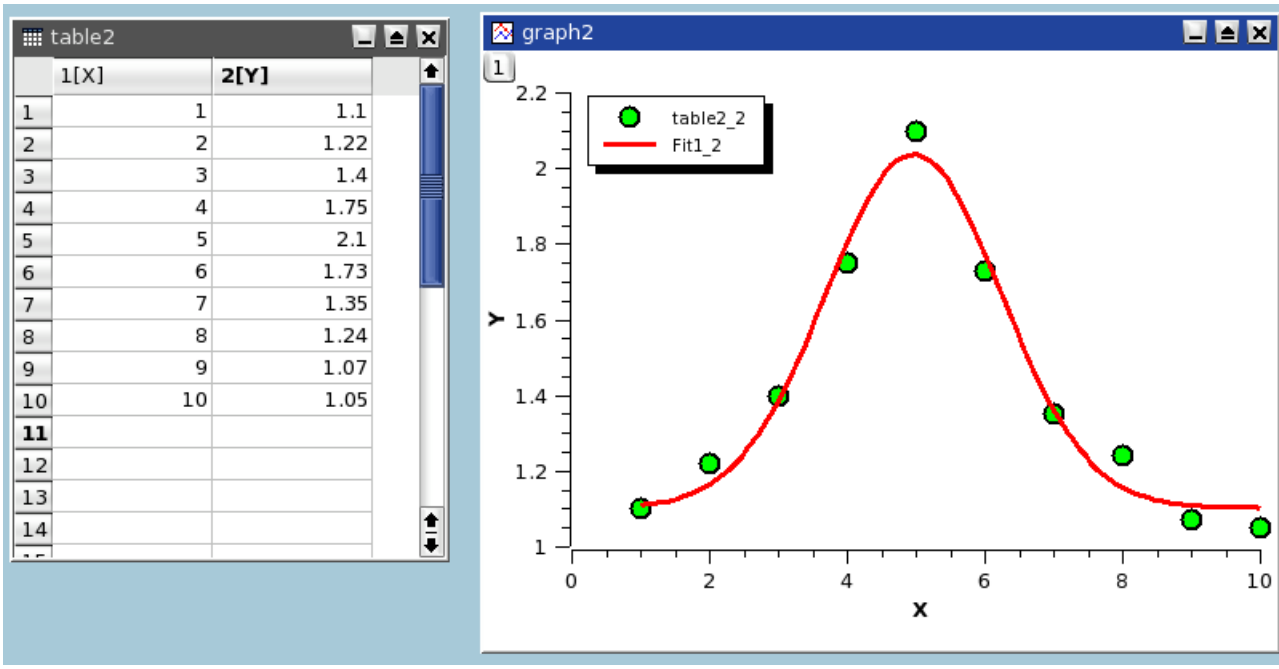


Figure 6.7: The results of a **Fit Gaussian**.

6.6.5 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 6.6.4: 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.

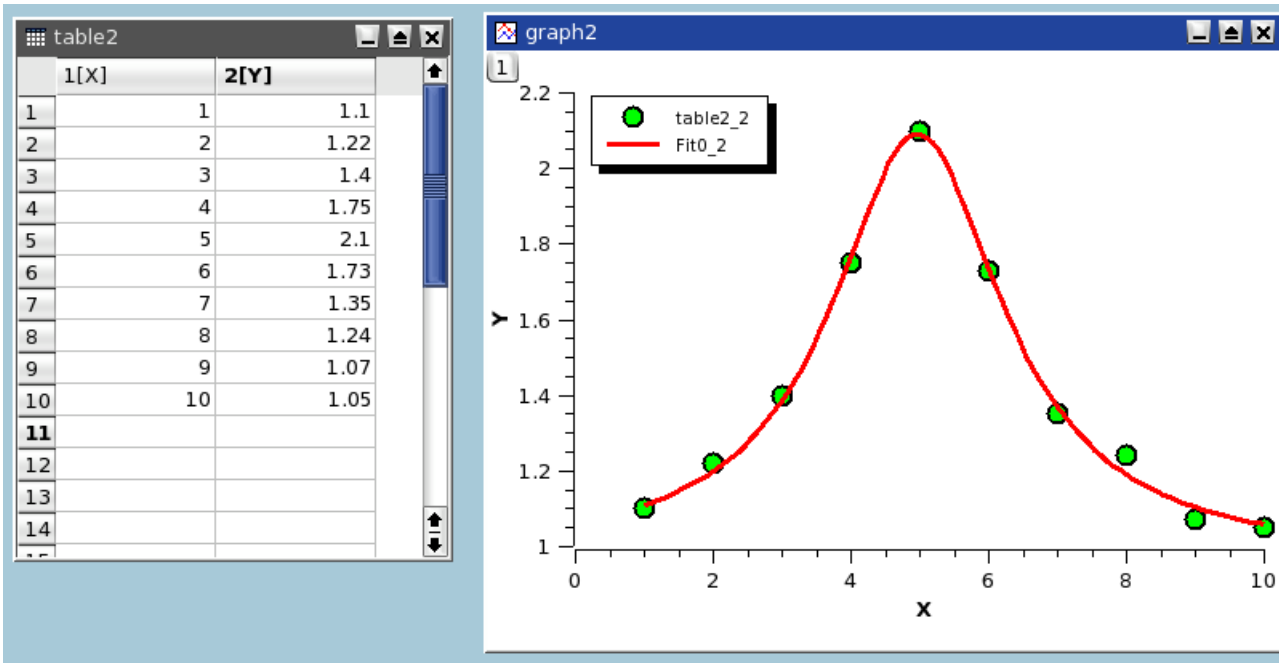


Figure 6.8: The results of a **Fit Lorentzian**.

6.7 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 clicking on the plot, then validate your choice for each peak with the *ENTER* key.

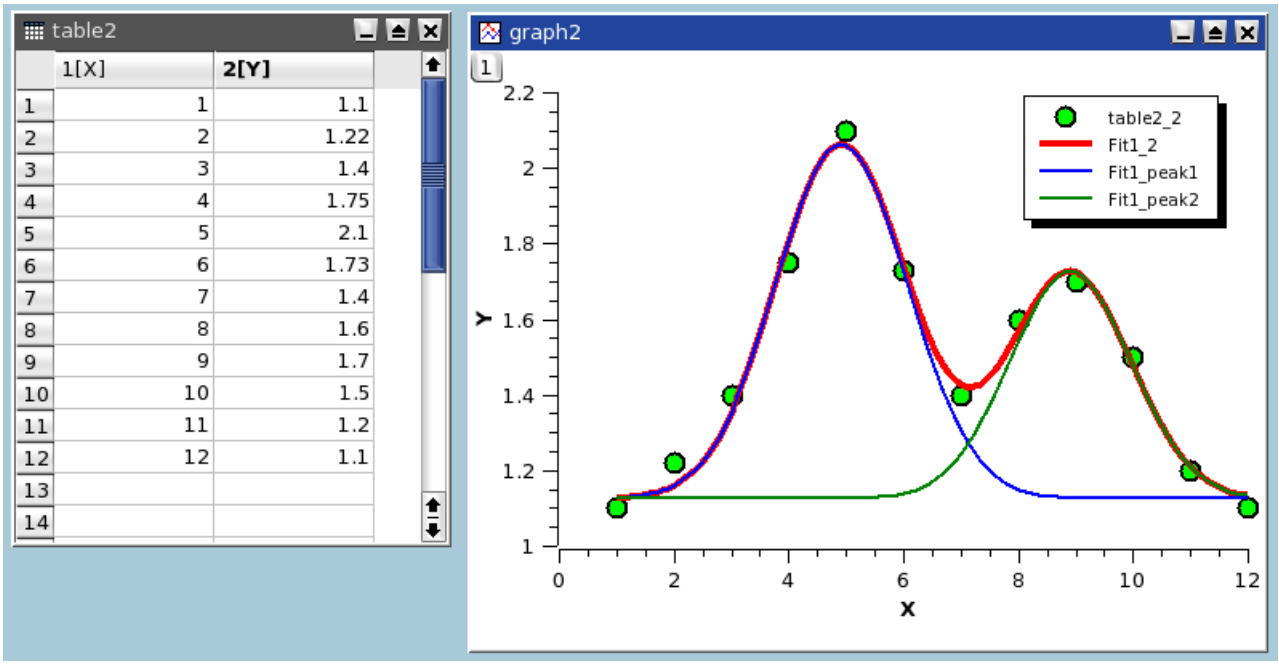
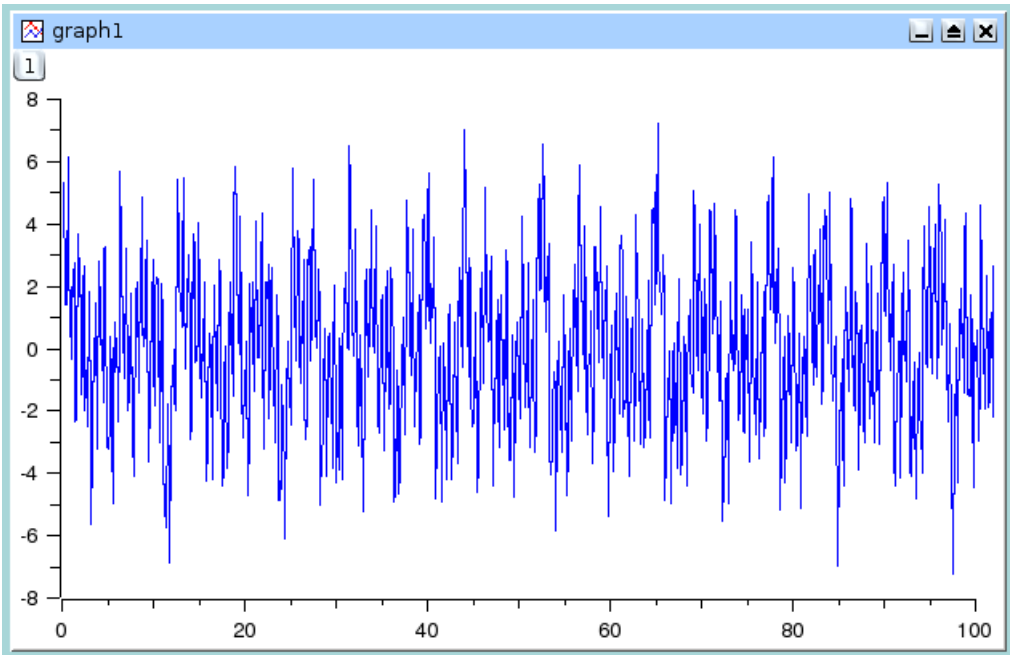


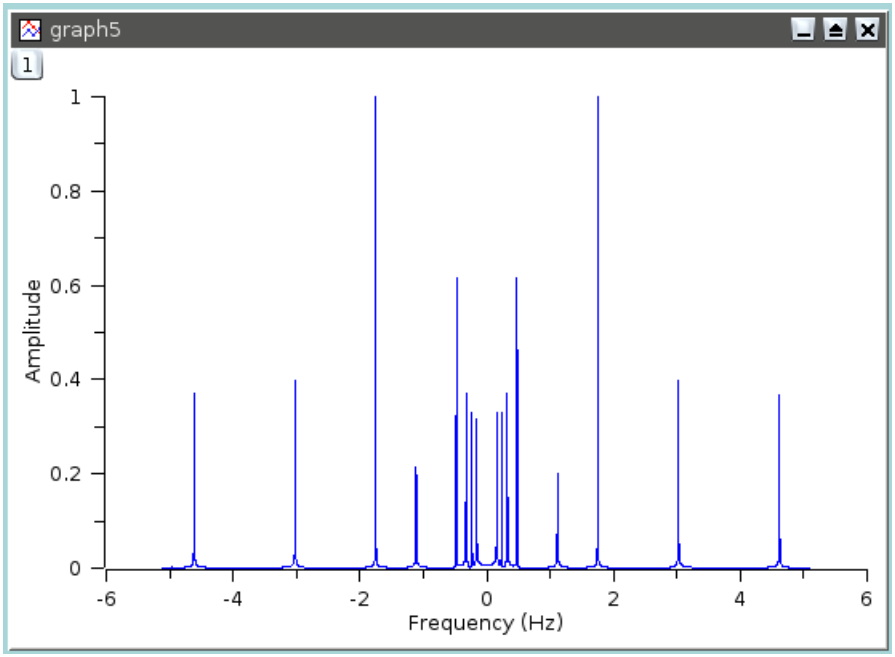
Figure 6.9: The results of a **Fit Multi-peak ->Gaussian...**.

6.8 Filtering of data curves

In this section, it will be assumed that you have the following data curve:



This signal has a power spectrum with high and low frequencies. We can analyze this by doing a FFT on the data curve, this leads to the following figure:



The newt sections will show the influence of the different filters on this data curve.

6.8.1 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 Hz, we will obtain:

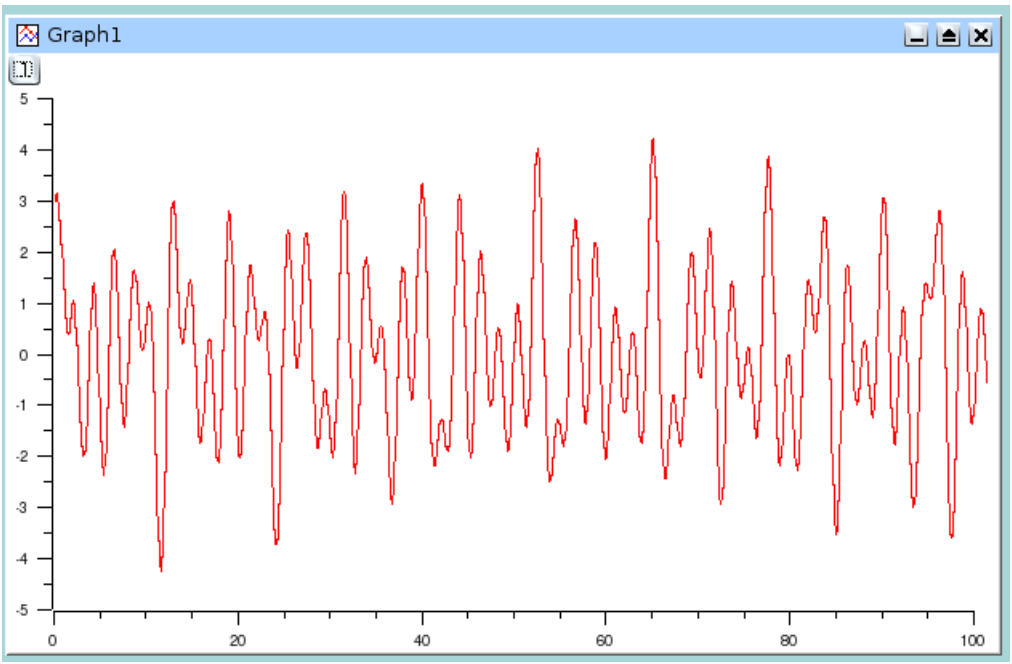
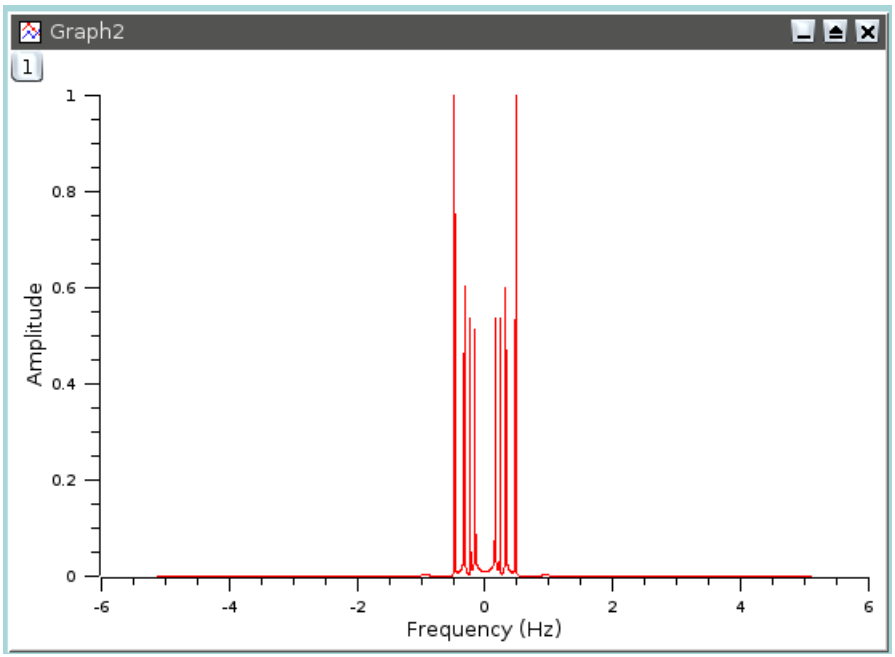


Figure 6.10: Signal after a FFT low pass filter

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



6.8.2 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 Hz, we will obtain:

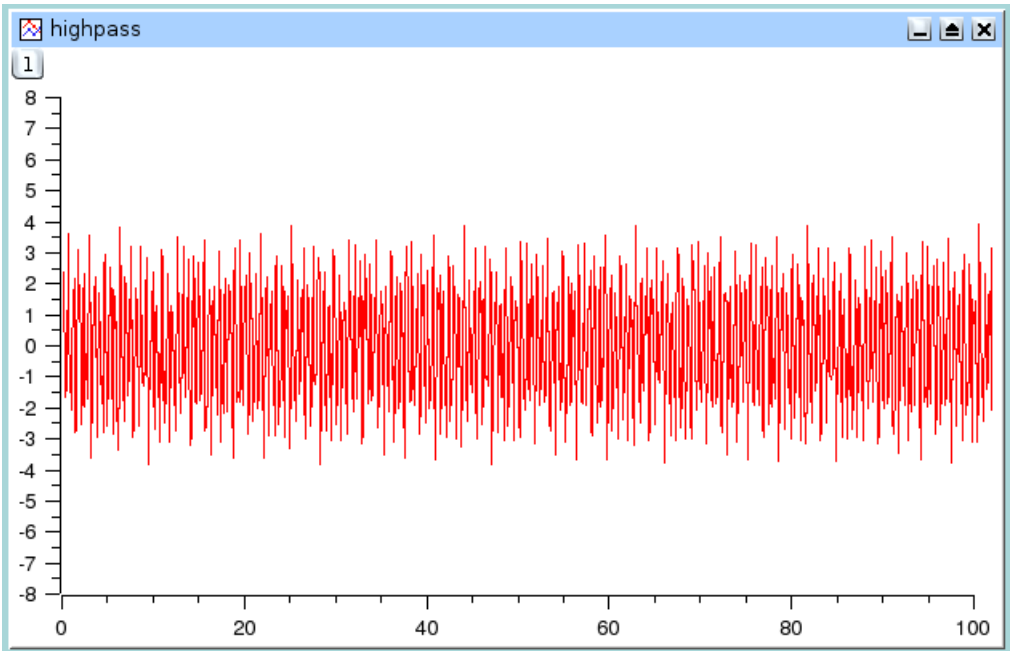
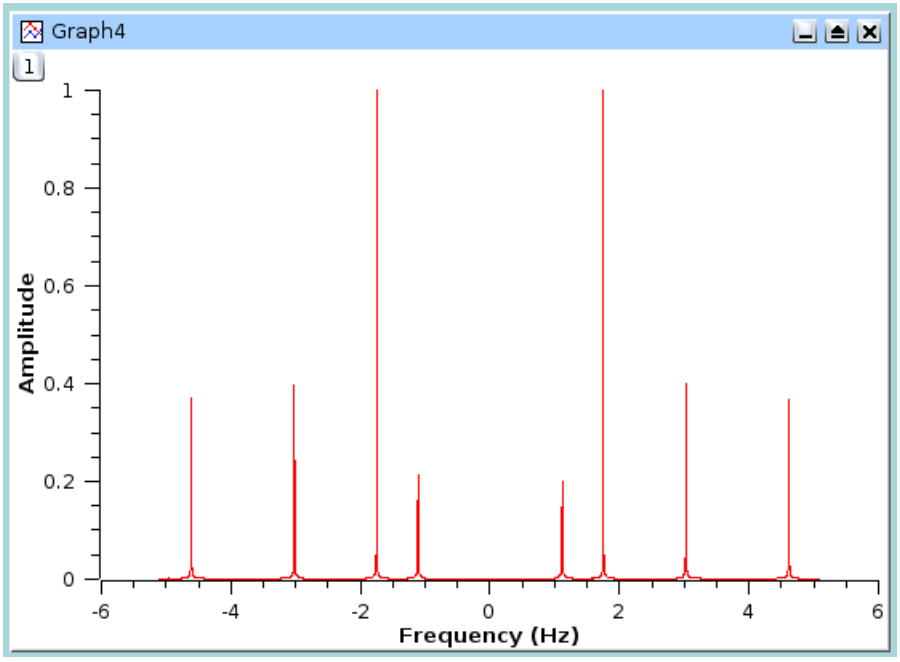


Figure 6.11: Signal after a FFT high pass filter

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



6.8.3 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:

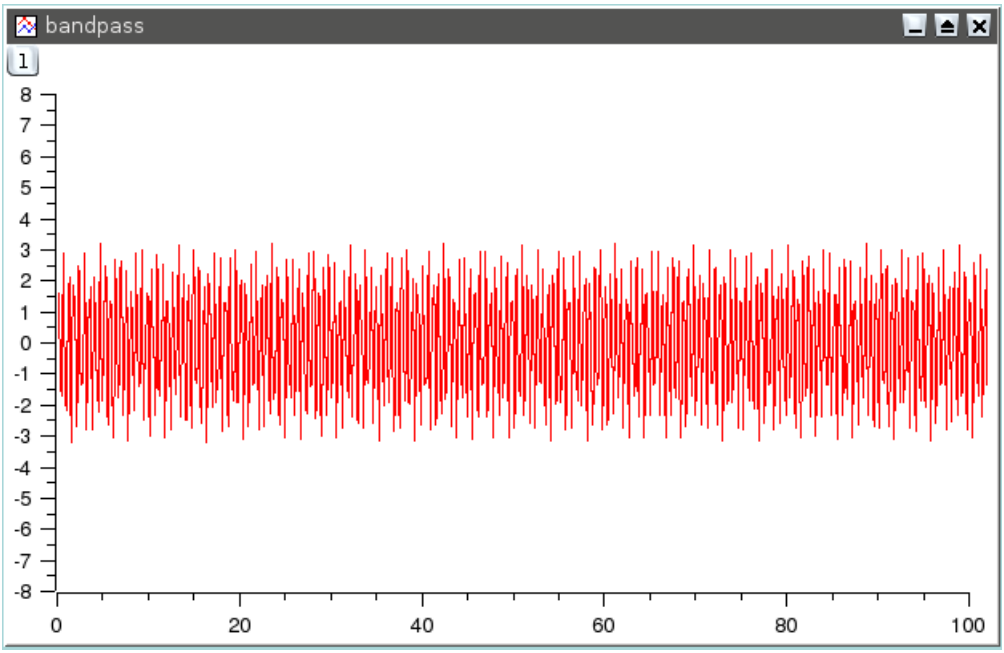
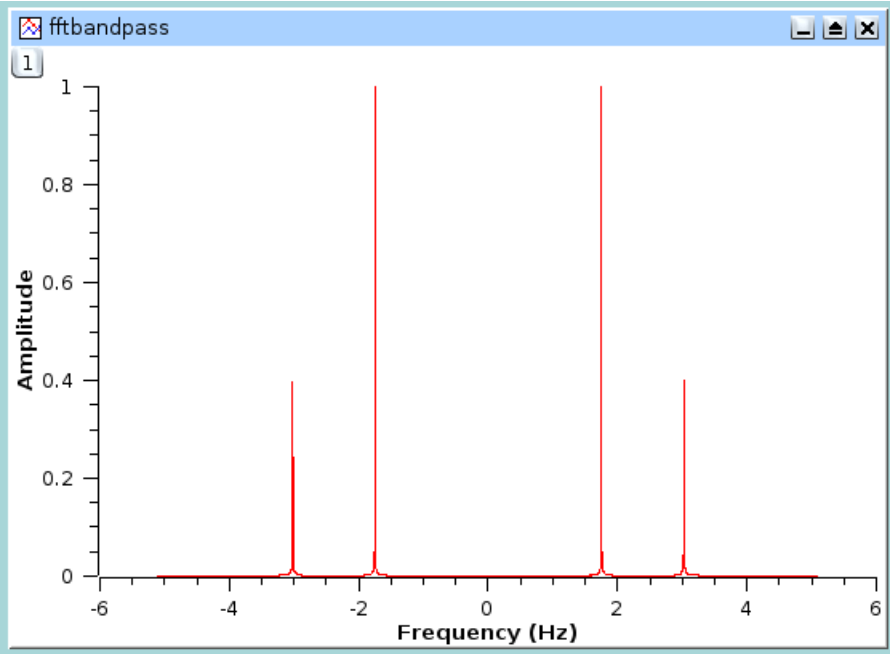


Figure 6.12: Signal after a FFT band pass filter

The power spectrum of this new signal shows that only the frequencies at 2 and 3 Hz have been kept.



6.8.4 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:

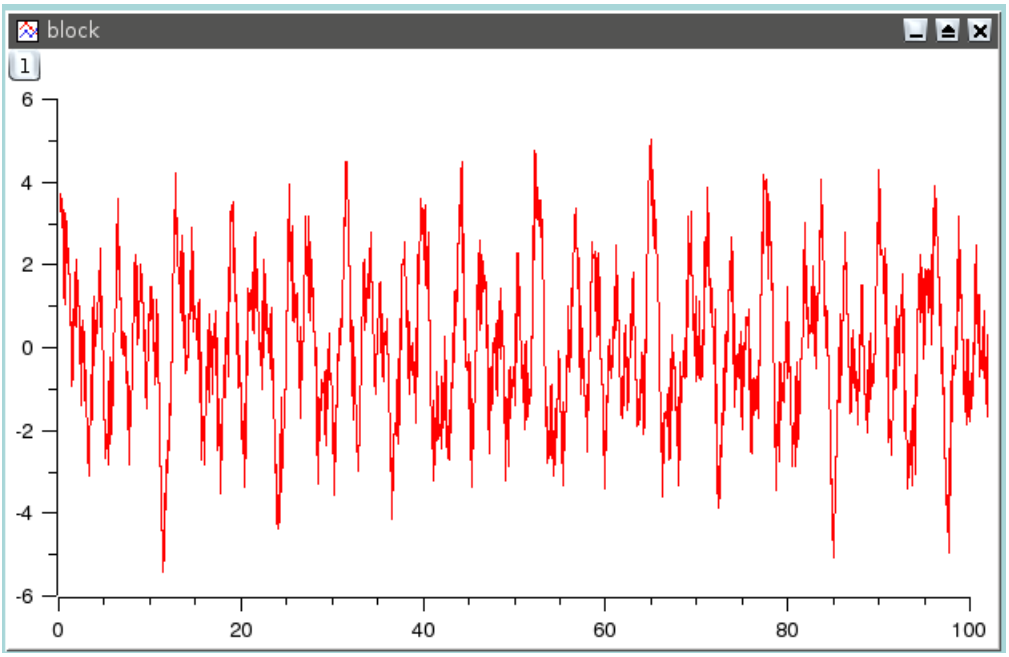
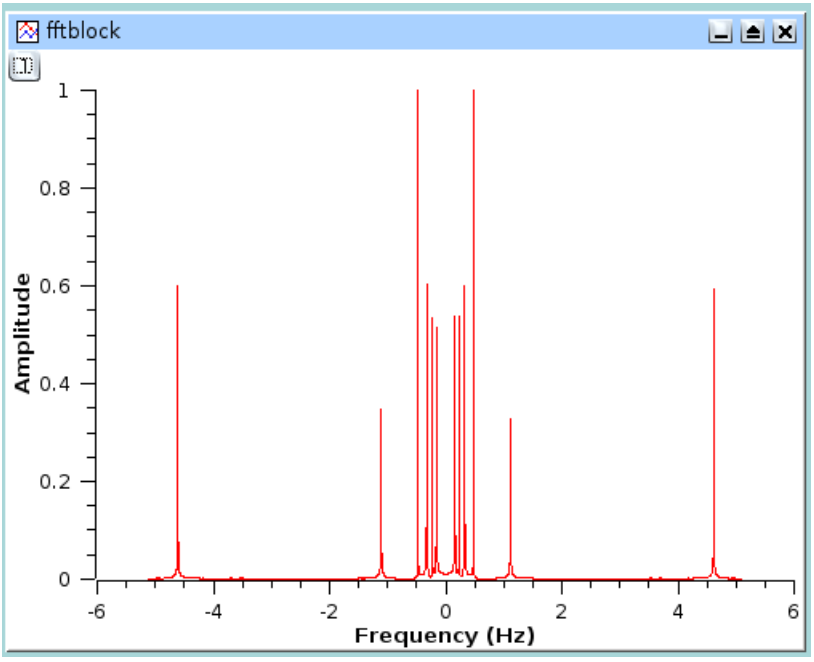


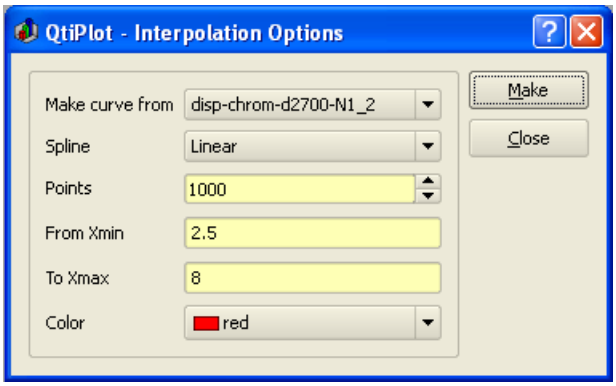
Figure 6.13: Signal after a FFT block band filter

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



6.9 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.

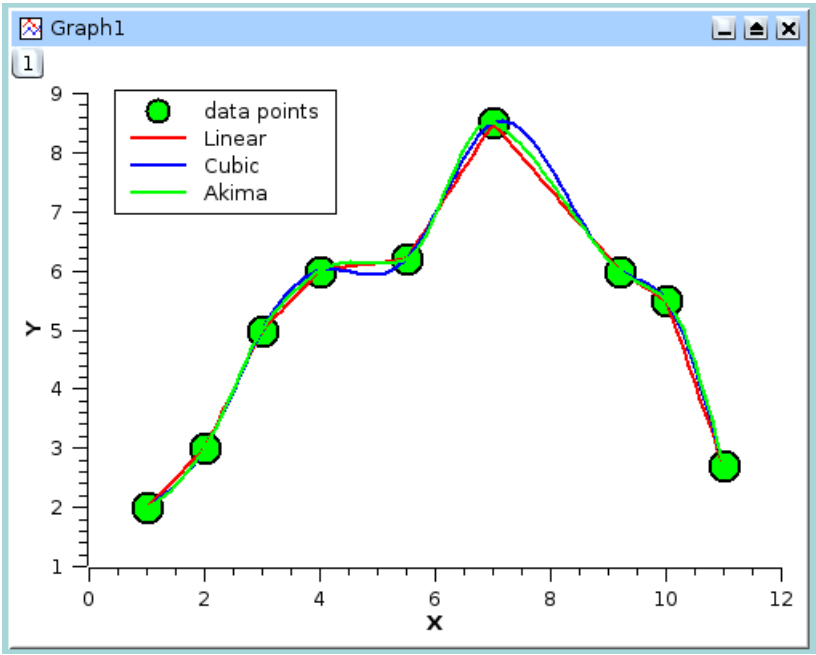


Figure 6.14: Comparison of the three methods of interpolation

Chapter 7

Mathematical Expressions and Scripting

QtiPlot supports two different interpreters for evaluating mathematical expressions and for executing scripts: *muParser* and *Python*. *muParser* can be only used for the evaluation of mathematical expressions, whereas *Python* can be used to execute scrips. The default interpreter is *muParser* therefore if you want to execute scripts you should first enable the *Python* scripting engine via the Scripting language dialog. Also, you can define the default scripting interpreter via the *General* tab of the Preferences dialog.

7.1 muParser

The constants _e=e=E and _pi=pi=PI=Pi are defined, as well as the following fundamental physical constants, operators and functions. Please note that the fundamental constants cannot be redefined. Doing so will raise an error message.

Name	Description
c	The speed of light in vacuum
eV	The energy of 1 electron volt
g	The standard gravitational acceleration on Earth
G	The gravitational constant
h	Planck's constant
hbar	Planck's constant divided by 2 pi
k	The Boltzmann constant
Na	Avogadro's number
R0	The molar gas constant
V0	The standard gas volume
Ry	The Rydberg constant, in units of energy

Table 7.1: Predefined Fundamental Physical Constants

Name	Description
+	Addition
-	Substraction
*	Multiplication
/	Division
^	Exponentiation (raise a to the power of b)
and	logical and (returns 0 or 1)
or	logical or (returns 0 or 1)
xor	logical exclusive or (returns 0 or 1)
<	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 7.2: Supported Mathematical Operators

7.2 Python

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

7.2.1 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 QtiPlot imports Python's standard math functions as well as (if available) special functions from SciPy, the symbolic mathematics library SymPy and helper functions for RPy2. Also, it creates some handy shortcuts, like `table("table1")` for `qti.app.table("table1")`.

When activating Python support, QtiPlot searches the initialization file in a default folder, which can be customized via the *File Locations* tab of the Preferences dialog. If the initialization file is not found, QtiPlot will issue a warning and *muParser* will be kept as the default interpreter.

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, QtiPlot will try to compile the source file (if you've got write permissions for the output file).

Name	Description
abs(x)	absolute value of x
acos(x)	inverse cosinus
acosh(x)	inverse hyperbolic cosinus
asin(x)	inverse sinus
asinh(x)	inverse hyperbolic sinus
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 n^{th} order, $J_n(x)$.
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 n^{th} 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.
cos(x)	cosinus of x
cosh(x)	hyperbolic cosinus 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.
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 logarythm of x
log(x)	decimal logarythm of x
log2(x)	base 2 logarythm of x
min(x1,x2,x3,...)	Minimum of the list of arguments
max(x1,x2,x3,...)	Maximum of the list of arguments
rint(x)	Round to nearest integer.
sign(x)	Sign function: -1 if x<0; 1 if x>0.
sin(x)	sinus of x
sinh(x)	hyperblic sinus of x
sqrt(x)	square root of x
tan(x)	tangent of x
tanh(x)	hyperbolic tangent of x

Table 7.3: Mathematical Functions

Name	Description
cell(a,b)	In the contex 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.
col(c)	Only works in the context of a table. Returns the value at column c and row i (the current row) in the context table. c can either be the column's number, or its name in doublequotes.
if(e1,e2,e3)	If e1 is true, e2 is executed else e3 is executed.
tablecol(t,c)	Only works in the context of a table. Returns the value at column c and row i (the current row) in the table t. t is the table's name in doublequotes, c is either the column's number or its name in doublequotes.

Table 7.4: Non-Mathematical Functions

7.2.2 Python Basics

Mathematical expressions work largely as expected. However, there's one caveat, especially when switching from muParser (which has been used exclusively in previous versions of QtiPlot): **`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
```

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,

```
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.

7.2.3 Defining Functions and Control Flow

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 QtiPlot'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
qti.mathFunctions["answer"] = answer
```

If you want to remove a function from the list, do:

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

Note that functions have their own local scope. That means that if you enter a function definition in a Note, you will not be able to access (neither reading nor writing) Note-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).

7.2.4 Mathematical Functions

Python comes with some basic mathematical functions that are automatically imported (if you use the initialization file shipped with QtiPlot). Along with them, the constants e (Euler's number) and pi (the one and only) are defined.

Name	Description
acos(x)	inverse cosinus
asin(x)	inverse sinus
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)	cosinus of x
cosh(x)	hyperbolic cosinus 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) logarythm of x
log10(x)	decimal (base 10) logarythm 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)	sinus of x
sinh(x)	hyperblic sinus of x
sqrt(x)	square root of x
tan(x)	tangent of x
tanh(x)	hyperbolic tangent of x

Table 7.5: Supported Mathematical Functions

7.2.5 Accessing QtiPlot's objects from Python

We will assume that you are using the initialization file shipped with QtiPlot. Accessing the objects in your project is straight-forward,

```
t = table("Table1")
m = matrix("Matrix1")
g = graph("Graph1")
p = plot3D("Graph2")
n = note("Notes1")
# get a pointer to the QTextEdit object used to display  ↩
    information in the results log window:
log = resultsLog()
```

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 a graph window named "test" with two layers disposed ↩
     on a 2 rows x 1 column grid
g = newGraph("test", 2, 2, 1)
# create an empty 3D plot window with default title
p = newPlot3D()
# create an empty note named "momo"
n = newNote("momo")
# use defaults
n = newNote()
```

The currently selected Table/Matrix etc. can be accessed with the follwing commands:

```
t = currentTable()
m = currentMatrix()
g = currentGraph()
n = currentNote()
```

The functions will only return a valid object if a window of the wanted type is actually selected. You can check if the object is valid with a simple if clause:

```
if isinstance(t,qti.Table): print "t is a table"
```

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")`**. Once you have established

contact with a MDI window, you can modify some of its properties, like the name, the window label, the geometry, etc.. For example, here's how to rename a window, change its label and the way they are displayed in the window title bar, the so called caption policy:

```
t = table("Table1")
setWindowName(t, "toto")
t.setWindowLabel("tutu")
t.setCaptionPolicy(MDIWindow.Both)
```

The caption policy can have one of the following values:

1. Name the window caption is determined by the window name
2. Labelthe caption is detemined by the window label
3. Bothcaption = "name - label"

You can access the name or label of a window by using the **`objectName()`** and **`windowLabel()`** functions. For a fast editing process, you can create template files from existing tables, matrices or plots. The templates can be used later on in order to create customized windows very easily:

```
saveAsTemplate(graph("Graph1"), "my_plot.qpt")
g = openTemplate("my_plot.qpt")
```

Also, you can easily clone a MDI window:

```
g1 = clone(graph("Graph1"))
```

If you want to delete a project window, you can use the **`close()`** method. You might want to deactivate the confirmation message, first:

```
w.confirmClose(False)
w.close()
```

All QtiPlot subwindows are displayed in a QMdiArea. You can get a pointer to this object via the **`workspace()`** method. This can be particularly usefull if you need to customize the behavior of the workspace via your scripts. Here follows a small example script that pops-up a message displaying the name of the active MDI subwindow each time a new window is activated:

```
def showMessage():
  QtGui.QMessageBox.about(qti.app, "", workspace(). ↩
      activeSubWindow().objectName())

QtCore.QObject.connect(workspace(), QtCore.SIGNAL(" ↩
    subWindowActivated(QMdiSubWindow *)"), showMessage)
```

7.2.6 Project Folders

Storing your data tables/matrices and your plots in folders can be very convenient and helpful when you're analysing loads of data files in the same project. New objects will always be added to the active folder. You can get a pointer to it via:

```
f = activeFolder()
```

The functions table, matrix, graph and note will start searching in the active folder and, failing this, will continue with a depth-first recursive search of the project's root folder, given by:

```
f = rootFolder()
```

In order to access subfolders and windows, there are the following functions:

```
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.

New folders can be created using:

```
newFolder = addFolder("New Folder", parentFolder = 0)
```

If the `parentFolder` is not specified, the new folder will be added as a subfolder of the project's root folder. When you create a new folder via a Python script, it doesn't automatically become the active folder of the project. You have to set this programatically, using:

```
changeFolder(newFolder, bool force=False)
```

Folders can be deleted using:

```
deleteFolder(folder)
```

You can save a folder as a project file, and of course, you can also save the whole project:

```
saveFolder(folder, "new_file.qti", compress=False)
saveProjectAs("new_file_2.qti", compress=False)
```

If `compress` is set to True, the project file will be archived to the .gz format, using zlib.

Also, you can load a QtiPlot or an Origin project file into a new folder. The new folder will have the base name of the project file and will be added as a subfolder to the `parentFolder` or to the current folder if no parent folder is specified.

```
newFolder = appendProject("projectName", parentFolder = 0)
```

If you don't want to be asked for confirmation when a table/matrix is renamed during this operation, or when deleting a folder via a Python script, you must change your preferences concerning prompting of warning messages, using the Preferences dialog ("Confirmations" tab).

Folders store their own log information containing the results of the analysis operations performed on the child windows. This information is updated in the result log window each time you change the active folder in the project. You can access and manipulate these log strings via the following functions:

```
text = folder.logInfo()
folder.appendLogInfo("Hello!")
folder.clearLogInfo()
```

7.2.7 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. In many places there is an alternative API which represents a table as a Python sequence is provided. Here rows are addressed by Python indices or slices which start at 0. These places are marked as such.

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)
```

An alternative way to get/set the value of a cell is using the format of the column (Text, Numeric, ...). Qtiplot handles all the casting under the hood and throws an `TypeError` if this isn't possible. Assigning `None` will clear the cell's value. The column type Day-of-Week returns/accepts the numbers 1 (monday) to 7 (sunday, for which also 0 is accepted). The column type Month returns/accepts the numbers 1 to 12. The column type Date returns/accepts `datetime.datetime` objects and also accepts a `QDateTime`. The column type Time returns/accepts `datetime.time` objects and also accepts a `QTime`.

```
t.cellData(col, row)
# and
t.setCellData(col, row, value)
```

The number of columns and rows is accessed via:

```
t.numRows() # same as len(t)
t.numCols()
t.setNumRows(number)
t.setNumCols(number)
```

You can add a new column at the end of the table or you can insert new columns before a `startColumn` using the functions bellow:

```
t.addColumn()
t.insertColumns(startColumn, count)
```

Adding an empty row at the end of the table is done with the `addRow()` method. It returns the new row number.

```
newRowIndex = t.addRow()
```

If you need all the data of a row or column you can use the `rowData()` and `colData()` methods. This is much faster then iterating manually over the cells. Alternatively you can use the `[]` operator in combination with Python indices or slices, which start at 0.

```
valueList = t.colData(col) # col may be a string or a number ↩
    starting at 1
rowTuple = t.rowData(row) # row number starting at 1
rowTuple = t[idx] # row index starts at 0
rowTupleList = t[slice]
```

A Table is iteratable. The data is returned row wise as tuple.

```
for c1, c2, c3 in t:
  # do stuff, assuming t has three columns
```

Assigning values to a complete row or column is also possible. While the new row data has to be a tuple which length must match the column number, column data just has to be iteratable. If the iterator stops before the end of the table is reached, a `StopIteration` exception is raised. In combination with the `offset` this allows to fill a column chunk wise. A positive offset starts filling the column after this row number. A negative offset ignores the firsts values of the iterator.

```
t.setColData(col, iterableValueSequence, offset=0)
# just fill the first column with a list of values, staring ↩
    at row 6
t.setColData(1, [12,23,34,56,67], 5)
# fill the second column with fibonacci numbers, omitting the ↩
     first three.
def FibonacciGenerator():
  a, b = 1, 1
  while True:
    a, b = b, a+b
    yield a
t.setColData(2, FibonacciGenerator(), -3)
```

```
t.setRowData(row, rowTuple) # row starts at 1
# assuming t has exactly two columns...
t.setRowData(2, (23, 5)) # fill the second row
t[1] = 23, 5 # using a Python index, starting at 0
# adding a new row and set it's values
t.appendRowData(rowTuple)
```

You can set the format of a column to text using:

```
t.setColTextFormat(col)
```

Or you can adjust the numeric format:

```
t.setColNumericFormat(col, format, precision, update=True)
```

were `col` is the number of the column to adjust and `precision` states the number of digits. The `format` can be one of the following:

Table.Default (0) standard format

Table.Decimal (1) decimal format with `precision` digits

Table.Scientific (2) scientific format

In the same way you can set a column hold a date. Here the text of a cell is interpreted using a format string:

```
t.setColDateFormat(col, format, update=True)
t.setColDateFormat("col1", "yyyy-MM-dd HH:mm")
```

were `col` is the name/number of a column and `format` the format string. In this string, the following placeholder are recognized:

d the day as number without a leading zero (1 to 31)

dd the day as number with a leading zero (01 to 31)

ddd the abbreviated localized day name (e.g. 'Mon' to 'Sun')

dddd the long localized day name (e.g. 'Monday' to 'Sunday')

M the month as number without a leading zero (1-12)

MM the month as number with a leading zero (01-12)

MMM the abbreviated localized month name (e.g. 'Jan' to 'Dec')

MMMM the long localized month name (e.g. 'January' to 'December')

yy the year as two digit number (00-99)

yyyy the year as four digit number

h the hour without a leading zero (0 to 23 or 1 to 12 if AM/PM display)

hh the hour with a leading zero (00 to 23 or 01 to 12 if AM/PM display)

H the hour without a leading zero (0 to 23, even with AM/PM display)

HH the hour with a leading zero (00 to 23, even with AM/PM display)

m the minute without a leading zero (0 to 59)

mm the minute with a leading zero (00 to 59)

s the second without a leading zero (0 to 59)

ss the second with a leading zero (00 to 59)

z the milliseconds without leading zeroes (0 to 999)

zzz the milliseconds with leading zeroes (000 to 999)

AP or A interpret as an AM/PM time. AP must be either "AM" or "PM".

ap or a interpret as an AM/PM time. ap must be either "am" or "pm".

Analog you can say that a text column should hold a time only...

```
t.setColTimeFormat(col, format, update=True)
t.setColTimeFormat(1, "HH:mm:ss")
```

... a month ...

```
t.setColMonthFormat(col, format, update=True)
t.setColMonthFormat(1, "M")
```

Here the format is the following:

M Only the first letter of the month, i.e. "J"

MMM The short form, like "Jan"

MMMM The full name, "January"

... or the day of week:

```
t.setColDayFormat(col, format, update=True)
t.setColDayFormat(1, "ddd")
```

Here the format is the following:

d Only the first letter of the day, i.e. "M"

ddd The short form, like "Mon"

dddd The full name, "Monday"

It is also possible to swap two columns using:

```
t.swapColumns(column1, column2)
```

You can delete a colum or a range of rows using the functions bellow:

```
t.removeCol(number)
t.deleteRows(startRowNumber, endRowNumber)
```

It is also possible to use Python's `del` statement to remove rows. Note that in this case a Python index or slice (instead of row numbers) is used, which start at 0.

```
del t[5] # deletes row 6
del t[0:4] # deletes row 1 to 5
```

Column names can be read and written with:

```
t.colName(number)
t.colNames()
t.setColName(col, newName, enumerateRight=False)
t.setColNames(newNamesList)
```

If `enumerateRight` is set to True, all the table columns starting from index `col` will have their names modified to a combination of the `newName` and a numerical increasing index. If this parameter is not specified, by default it is set to False. The plural forms get/set all headers at once.

You can change the plot role of a table column (abscissae, ordinates, error bars, etc...) using:

```
t.setColumnRole(col, role)
```

where `role` specifies the desired column role:

0. Table.None

1. Table.X

2. Table.Y

3. Table.Z

4. Table.xErr

5. Table.yErr

6. Table.Label

You can normalize a single column or all columns in a table:

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

Sort a single or all columns:

```
t.sortColumn(col, order = 0)
t.sort(type = 0, order = 0, leadingColumnName)
```

7.2.7.1 Import ASCII files

Import values from `file`, using `sep` as separator char, ignoring `ignoreLines` lines at the head of the file and all lines starting with a `comment` string.

```
t.importASCII(file, sep="\t",ignoreLines=0,renameCols=False, ↩
    stripSpaces=True,simplifySpace=False,
importComments=False,comment="#",readOnly=False,importAs= ↩
    Table.Overwrite,locale=QLocale(),endLine=0,maxRows=-1)
```

As you see from the above list of import options, you have the possibility to set the new columns as read-only. This will prevent the imported data from beeing modified. You have the possibility to remove this protection at any time, by using:

```
t.setReadOnlyColumn(col, False)
```

The `importAs` flag can have the following values:

0. Table.NewColumns: data values are added as new columns.

1. Table.NewRows: data values are added as new rows.

2. Table.Overwrite: all existing values are overwritten (default value).

The `endLine` flag specifies the end line character convention used in the ascii file. Possible values are: 0 for line feed (LF), which is the default value, 1 for carriage return + line feed (CRLF) and 2 for carriage return only (usually on Mac computers).

The last parameter `maxRows` allows you to specify a maximum number of imported lines. Negative values mean that all data lines must be imported.

If the decimal separator of the imported file does not match the currently used conventions, you have to adjust them before using the table:

```
t.setDecimalSeparators(country,ignoreGroupSeparator=True)
```

Where country can have one of the following values:

0. Use the system value

1. Use the following format: 1,000.00

2. Use the following format: 1.000,00

3. Use the following format: 1 000,00

7.2.7.2 Importing Excel sheets

It is possible to import a sheet from an Excel .xls file `file` to a table, using:

```
t = importExcel(file, sheet)
```

If the integer `sheet` variable is not specified, all non-empty sheets in the Excel workbook are imported into separate tables and a reference to the table containing the data from the last sheet is returned.

7.2.7.3 Importing ODF spreadsheets

It is possible to import a sheet from an ODF spreadsheet .ods `file` to a table, using:

```
t = importOdfSpreadsheet(file, sheet)
```

If the integer `sheet` variable is not specified, all non-empty sheets in the spreadsheet are imported into separate tables and a reference to the table containing the data from the last sheet is returned.

7.2.7.4 Export Tables

You can export values from a table to an ASCII `file`, using `sep` as separator charar-acter. The `ColumnLabels` option allows you to export or ignore the column labels, `ColumnComments` does the same for the comments displayed in the table header and the `SelectionOnly` option makes possible to export only the selected cells of the table.

```
t.exportASCII(file,sep="\t",ignore=0,ColumnLabels=False, ↩
    ColumnComments=False,SelectionOnly=False)
```

Other settings that you can modify are the text displayed as a comment in the header of a column...

```
t.setComment(col, newComment)
```

... or the expression used to calculate the column values. Please beware that changing the command doesn't automatically update the values of the column; you have to call `recalculate` explicitly. Calling it with just the column as argument will recalculate every row. Forcing muParser can speed things up.

```
t.setCommand(col, newExpression)
t.recalculate(col, startRow=1, endRow=-1, forceMuParser=False ↩
    , notifyChanges=True)
```

You can access the column comments and enable/disable their display via the following functions:

```
t.comment(col)
t.showComments(on = True)
```

You can also modify the width of a column (in pixels) or hide/show table columns:

```
t.setColumnWidth(col, width)
t.hideColumn(col, True)
```

If one or several tabel columns are hidden you can make them visible again using:

```
t.showAllColumns()
```

You can ensure the visibility of a cell with:

```
t.scrollToCell(col, row)
```

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()
```

While the above is easy to understand, there is a faster and more pythonic way of doing the same:

```
t = table("table1")
t.setColData(1, [i*i for i in range(len(t))])
t.notifyChanges()
```

You can check if a column or row of a table is selected by using the following functions:

```
t.isColSelected(col)
t.isRowSelected(row)
```

7.2.7.5 R interface

If RPy2 is available, the default initialization file sets up the helper functions **`qti.Table.toRDataFrame`** and **`qti.app.newTableFromRDataFrame`** to convert back and forth between R data frames and QtiPlot tables. Here is a little example of an R session...

```
df <- read.table("/some/path/data.csv", header=TRUE)
m <- mean(df)
v <- var(df)
source("/some/path/my_func.r")
new_df <- my_func(df, foo=bar)
```

... and now the same from within QtiPlot:

```
df = table("Table1").toRDataFrame()
print R.mean(df), R.var(df)
R.source("/some/path/my_func.r")
new_df = R.my_func(df, foo=bar)
newTableFromRDataFrame(new_df, "my result table")
```

7.2.8 Working with Matrices

Matrix objects have a dual view mode: either as images or as data tables. Assuming that you have assigned some matrix to the variable `m`, you can change its display mode via the following function:

```
m.setViewType(Matrix.TableView)
m.setViewType(Matrix.ImageView)
```

If a matrix is viewed as an image, you have the choice to display it either as gray scale or using a predefined color map:

```
m.setGrayScale()
m.setRainbowColorMap()
m.setDefaultColorMap() # default color map defined via the 3D ↩
     plots tab of the preferences dialog
```

You can also define custom color maps:

```
map = LinearColorMap(QtCore.Qt.yellow, QtCore.Qt.blue)
map.setMode(LinearColorMap.FixedColors) # default mode is  ↩
    LinearColorMap.ScaledColors
map.addColorStop(0.2, QtCore.Qt.magenta)
map.addColorStop(0.7, QtCore.Qt.cyan)
m.setColorMap(map)
```

You have direct access to the color map used for a matrix via the following functions:

```
map = m.colorMap()
col1 = map.color1()
print col1.green()
col2 = map.color2()
print col2.blue()
```

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)
```

An alternative solution to assign values to a Matrix, would be to define a formula and to calculate the values using this formula, like in the following example:

```
m.setFormula("x*y*sin(x*y)")
m.calculate()
```

You can also specify a column/row range in the calculate() function, like this:

```
m.calculate(startRow, endRow, startColumn, endColumn)
```

Before setting the values in a matrix you might want to define the numeric precision, that is the number of significant digits used for the computations:

```
m.setNumericPrecision(prec)
```

Also, like with tables, you can access the number of rows/columns in a matrix:

```
rows = m.numRows()
columns = m.numCols()
```

Matrix objects allow you to define a system of x/y coordinates that will be used when plotting color/contour maps or 3D height maps. You can manipulate these coordinates using the following functions:

```
xs = m.xStart()
xe = m.xEnd()
ys = m.yStart()
ye = m.yEnd()
m.setCoordinates(xs + 2.5, xe, ys - 1, ye + 1)
```

The horizontal and vertical headers of a matrix can display either the x/y coordinates or the column/row indexes:

```
m.setHeaderViewType(Matrix.ColumnRow)
m.setHeaderViewType(Matrix.XY)
```

There are several built-in transformations that you can apply to a matrix object. You can transpose or invert a matrix and calculate its determinant, provided, of course, that the conditions on the matrix dimensions, required by these operations, are matched:

```
m.transpose()
m.invert()
d = m.determinant()
```

Some other operations, very useful when working with images, like 90 degrees rotations and mirroring, can also be performed. By default rotations are performed clockwise. For a counterclockwise rotation you must set the `clockwise` parameter to `False`.

```
m.flipVertically()
m.flipHorizontally()
m.rotate90(clockwise = True)
```

Please note that sometimes, after a change in the matrix settings, you need to use the following function in order to update the display:

```
m.resetView()
```

If you need to get data from a table, in order to use it in a matrix (or vice-versa), you can avoid time consuming copy/paste operations and speed up the whole proces by simply converting the table into a matrix:

```
m = tableToMatrix(table("Table1"))
t = matrixToTable(m)
```

Also, it's worth knowing that you can easily import image files to matrices, that can be used afterwards for plotting (see the next section for more details about 2D plots):

```
m1 = importImage("C:/poze/adi/PIC00074.jpg")
m2 = newMatrix()
m2.importImage("C:/poze/adi/PIC00075.jpg")
```

The algorithm used to import the image returns a gray value between 0 and 255 from the (r, g, b) triplet corresponding to each pixel. The gray value is calculated using the formula: (r * 11 + g * 16 + b * 5)/32

For custom image analysis operations, you can get a copy of the matrix image view, as a QImage object, via:

```
image = m.image()
```

You can export matrices to all raster image formats supported by Qt or to any of the following vectorial image format: EPS, PS, PDF or SVG using:

```
m.export(fileName)
```

This is a shortcut function which uses some default parameters in order to generate the output image. If you need more control over the export parameters you must use one of the following functions:

```
m1.exportRasterImage(fileName, quality = 100, dpi = 0)
m2.exportVector(fileName, resolution, color = True)
```

, where the `quality` parameter influences the size of the output file. The higher this value (maximum is 100), the higher the qualitity of the image, but the larger the size of the resulting files. The `dpi` parameter represents the export resolution in pixels per inch (the default is screen resolution).

You can also import an ASCII data `file`, using `sep` as separator characters, ignoring `ignore` lines at the head of the file and all lines starting with a `comment` string:

```
m.importASCII(file, sep="\t", ignore=0, stripSpaces=True,  ↩
    simplifySpace=False, comment="#",
        importAs=Matrix.Overwrite, locale=QLocale(), endLine ↩
            =0, maxRows=-1)
```

The `importAs` flag can have the following values:

0. Matrix.NewColumns: data values are added as new columns.

1. Matrix.NewRows: data values are added as new rows.

2. Matrix.Overwrite: all existing values are overwritten (default value).

The `locale` parameter can be used to specify the convention for decimal separators used in your ASCII file.

The `endLine` flag specifies the end line character convention used in the ascii file. Possible values are: 0 for line feed (LF), which is the default value, 1 for carriage return + line feed (CRLF) and 2 for carriage return only (usually on Mac computers).

The last parameter `maxRows` allows you to specify a maximum number of imported lines. Negative values mean that all data lines must be imported.

Also, you can export values from a matrix to an ASCII `file`, using `sep` as separator chararacters. The `SelectionOnly` option makes possible to export only the selected cells of the matrix.

```
m.exportASCII(file, sep="\t", SelectionOnly=False)
```

7.2.9 Stem Plots

A stemplot (or stem-and-leaf plot), in statistics, is a device for presenting quantitative data in a graphical format, similar to a histogram, to assist in visualizing the shape of a distribution. A basic stemplot contains two columns separated by a vertical line. The left column contains the stems and the right column contains the leaves. See Wikipedia for more details.

QtiPlot provides a text representation of a stemplot. The following function returns a string of characters reprezenting the statistical analysis of the data:

```
text = stemPlot(Table *t, columnName, power = 1001, startRow ↩
    = 0, endRow = -1)
```

where the `power` variable is used to specify the stem unit as a power of 10. If this parameter is greater than 1000 (the default behavior), than QtiPlot will try to guess the stem unit from the input data and will pop-up a dialog asking you to confirm the automatically detected stem unit.

Once you have created the string representation of the stemplot, you can display it in any text editor you like: in a note within the project or even in the results log:

```
resultsLog().append(stemPlot(table("Table1"), "Table1_2", 1, ↩
    2, 15))
```

7.2.10 2D Plots

If you want to create a new Graph window for some data in table Table1, you can use the plot command:

```
t = table("Table1")
g = plot(t, column, type)
```

`type` specifies the desired plot type and can be one of the following numbers or the equivalent reserved word:

0 Layer.Line

1 Layer.Scatter

2 Layer.LineSymbols

3 Layer.VerticalBars

4 Layer.Area

5 Layer.Pie

6 Layer.VerticalDropLines

7 Layer.Spline

8 Layer.HorizontalSteps

9 Layer.Histogram

10 Layer.HorizontalBars

13 Layer.Box

15 Layer.VerticalSteps

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

```
g1 = plot(table("Table1"), (2,4,7), 2)
g2 = plot(table("Table1"), ("Table1_2","Table1_3"), Layer. ↩
    LineSymbols)
```

All the curves in a plot layer can be customized in terms of color, line width and line style. Here's a short script showing the corresponding functions at work:

```
t = newTable("test", 30, 4)
for i in range(1, t.numRows()+1):
  t.setCell(1, i, i)
  t.setCell(2, i, i)
  t.setCell(3, i, i+2)
  t.setCell(4, i, i+4)

l = plot(t, (2,3,4), Layer.Line).activeLayer() # plot columns ↩
     2, 3 and 4
for i in range(0, l.numCurves()):
  l.setCurveLineColor(i, 1 + i) #curve color is defined as an ↩
       integer value
  l.setCurveLineWidth(i, 0.5 + 2*i)

l.setCurveLineStyle(1, QtCore.Qt.DotLine)
l.setCurveLineStyle(2, QtCore.Qt.DashLine)
```

You can also create a vector plot by giving four columns in a Python tuple as an argument and the plot type as Layer.VectXYXY (11) or Layer.VectXYAM (14), depending on how you want to define the end point of your vectors: using (X, Y) coordinates or (Angle, Magnitude) coordinates.

```
g = plot(table("Table1"), (2,3,4,5), Layer.VectXYXY)
```

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)
```

7.2.10.1 Working with curves

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

```
l.insertCurve(table, Ycolumn, type=Layer.Scatter, int startRow = 0, int endRow = -1)# returns a reference to the inserted curve
l.insertCurve(table, Xcolumn, Ycolumn, type=Layer.Scatter, int startRow = 0, int endRow = -1)# returns a reference to the inserted curve
l.addCurve(table, column, type=Layer.Line, lineWidth = 1, symbolSize = 3, startRow = 0, endRow = -1)# returns True on success
l.addCurves(table, (2,4), type=Layer.Line, lineWidth = 1, symbolSize = 3, startRow = 0, endRow = -1)# returns True on success
l.removeCurve(curveName)
l.removeCurve(curveIndex)
l.removeCurve(curveReference)
l.deleteFitCurves()
```

It is possible to change the order of the curves inserted in a layer using the following function:

```
l.changeCurveIndex(int oldIndex, int newIndex)
```

Sometimes, when performing data analysis, one might need the curve title. It is possible to obtain it using the method bellow:

```
title = l.curveTitle(curveIndex)
```

It is possible to get a reference to a curve on the layer l using it's index or it's title, like shown bellow:

```
c = l.curve(curveIndex)
c = l.curve(curveTitle)
dc = l.dataCurve(curveIndex)
```

Please, keep in mind the fact that the above methods might return an invalid reference if the curve with the specified index/title is not a PlotCurve or a DataCurve object, respectively. For example, an analytical function curve is a PlotCurve but not

a DataCurve and spectrograms are a completely different type of plot items which are neither PlotCurves nor DataCurves.

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

```
l.setCurveAxes(number, x-axis, y-axis)
```

where number is the curve's number, x-axis is either 0 or 1 (bottom or top) and y-axis is either 0 or 1 (left or right). You can also add analytical function curves to a plot layer:

```
c = l.addFunction("x*sin(x)", 0, 3*pi, points = 100)
c.setTitle("x*sin(x)")
c.setPen(Qt.green)
c.setBrush(QtGui.QColor(0, 255, 0, 100))

l.addParametricFunction("cos(m)", "sin(m)", 0, 2*pi, points = ↩
     100, variableName = "m")
l.addPolarFunction("t", "t", 0, 2*pi, points = 100,  ↩
    variableName = "t")
```

When dealing with analytical function curves, you can customize them using the following methods:

```
c.setRange(0, 2*pi)
c.setVariable("t")
c.setFormulas("sin(t)", "cos(t)")
c.setFunctionType(FunctionCurve.Polar) # or c.setFunctionType ↩
    (FunctionCurve.Parametric)
c.loadData(1000, xLog10Scale = False)

c.setFunctionType(FunctionCurve.Normal)
c.setFormula("cos(x)")
c.loadData()
```

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

```
l.numCurves()
```

Once you have added a curve to a 2D plot, you can fully customize it's appearance:

```
l = newGraph().activeLayer()
l.setAntialiasing()
c = l.insertCurve(table("Table1"), "Table1_2", Layer. ↩
    LineSymbols)
c.setPen(QtGui.QPen(Qt.red, 3))
c.setBrush(QtGui.QBrush(Qt.darkYellow))
c.setSymbol(PlotSymbol(PlotSymbol.Hexagon, QtGui.QBrush(Qt. ↩
    yellow), QtGui.QPen(Qt.blue, 1.5), QtCore.QSize(15, 15)))
```

It is possible to change the number of symbols to be displayed for a curve using the function bellow. This option can be very usefull for very large data sets:

```
c.setSkipSymbolsCount(3)
print c.skipSymbolsCount()
```

An alternative way of customizing a curve is by using the functions bellow:

```
l.setCurveLineColor(int curve, int color) # uses the index of ↩
     the colors in the default QtiPlot color list: 0 = black, ↩
     1 = red, 2 = green, etc...
l.setCurveLineStyle(int curve, Qt::PenStyle style)
l.setCurveLineWidth(int curve, double width)
```

You can also define a global color policy for the plot layer using the following convenience functions:

```
l.setGrayScale()
l.setIndexedColors() # uses the colors in the default QtiPlot ↩
     color list: 0 = black, 1 = red, 2 = green, etc...
```

You can display labels showing the y values for each data point in a DataCurve:

```
c.setLabelsColumnName("Table1_2")
c.setLabelsOffset(50, 50)
c.setLabelsColor(Qt.red)
c.setLabelsFont(QtGui.QFont("Arial", 14))
c.setLabelsRotation(45)
c.loadData() # creates the labels and updates the display
```

and, of course, you can disable them using:

```
c.clearLabels()
l.replot() # redraw the plot layer object
```

If you need to change the range of data points displayed in a DataCurve you can use the following methods:

```
c.setRowRange(int startRow, int endRow)
c.setFullRange()
```

Also, you can hide/show a plot curve via:

```
c.setVisible(bool on)
```

In case you need to get information about the data stored in the curve, you have at your disposal the functions bellow:

```
points = c.dataSize()
for i in range (0, points):
  print i, "x = ", c.x(i), "y = ", c.y(i)

print c.minXValue()
print c.maxXValue()
print c.minYValue()
print c.maxYValue()
```

7.2.10.2 Plot symbols

Here's how you can customize the plot symbol used for a 2D plot curve `c`:

```
s = c.symbol()
s.setSize(QtCore.QSize(7, 7))# or s.setSize(7)
s.setBrush(QtGui.QBrush(Qt.darkYellow))
s.setPen(QtGui.QPen(Qt.blue, 3))
s.setStyle(PlotSymbol.Diamond)
l.replot() # redraw the plot layer object
```

The symbol styles available in QtiPlot are:

0 PlotSymbol.NoSymbol

1 PlotSymbol.Ellipse

2 PlotSymbol.Rect

3 PlotSymbol.Diamond

4 PlotSymbol.Triangle

5 PlotSymbol.DTriangle

6 PlotSymbol.UTriangle

7 PlotSymbol.LTriangle

8 PlotSymbol.RTriangle

9 PlotSymbol.Cross

10 PlotSymbol.XCross

11 PlotSymbol.HLine

12 PlotSymbol.VLine

13 PlotSymbol.Star1

14 PlotSymbol.Star2

15 PlotSymbol.Hexagon

7.2.10.3 Image and Contour Line Plots (Spectrograms)

As you have seen in the previous section, it is possible create 2D plots from matrices. Here's how you can do it in practice:

```
m = importImage("C:/poze/adi/PIC00074.jpg")
g1 = plot(m, Layer.ColorMap)
g2 = plot(m, Layer.Contour)
g3 = plot(m, Layer.GrayScale)
```

The plot functions above return a reference to the multilayer plot window. If you need a reference to the spectrogram object itself, you can get it as shown in the example bellow:

```
m = newMatrix("TestMatrix", 1000, 800)
m.setFormula("x*y")
m.calculate()
g = plot(m, Layer.ColorMap)
s = g.activeLayer().spectrogram(m)
s.setColorBarWidth(20)
```

It is possible to fine tune the plots created from a matrix:

```
m = newMatrix("TestMatrix", 1000, 800)
m.setFormula("x*y")
m.calculate()

s = newGraph().activeLayer().plotSpectrogram(m, Layer. ↩
    ColorMap)
s.setContourLevels((20.0, 30.0, 60.0, 80.0))
s.setDefaultContourPen(QtGui.QPen(Qt.yellow)) # set global  ↩
    pen for the contour lines
s.setLabelsWhiteOut(True)
s.setLabelsColor(Qt.red)
s.setLabelsFont(QtGui.QFont("Arial", 14))
s.setLabelsRotation(45)
s.showColorScale(Layer.Top)
s.setColorBarWidth(20)
```

As you have seen earlier, you can set a global pen for the contour lines, using:

```
s.setDefaultContourPen(QtGui.QPen(Qt.yellow))
```

You can also assign a specific pen for each contour line, using the function bellow:

```
s.setContourLinePen(index, QPen)
```

or you can automatically set pen colors defined by the color map of the spectrogram:

```
s.setColorMapPen(bool on = True)
```

You can also use any of the following functions:

```
s.setMatrix(Matrix *, bool useFormula = False)
s.setUseMatrixFormula(bool useFormula = True)# calculate data ↩
     to be drawn using matrix formula (if any)
s.setLevelsNumber(int)
s.showColorScale(int axis, bool on = True)
s.setGrayScale()
s.setDefaultColorMap()
s.setCustomColorMap(LinearColorMap map)
s.showContourLineLabels(bool show = True) # enable/disable  ↩
    contour line labels
```

```
s.setLabelsOffset(int x, int y) # offset values for all  ↩
    labels in % of the text size
s.updateData()
```

7.2.10.4 Histograms

As you have seen in the previous section, it is possible create 2D histograms from matrices or tables. Here's a small script showing how to customize a histogram and how to get access to the statistical information in the histogram (bin positions, counts, mean, standard deviation, etc...):

```
m = newMatrix("TestHistogram", 1000, 800)
m.setFormula("x*y")
m.calculate()

g = newGraph().activeLayer()
h = g.addHistogram(m)
h.setBinning(10, 1, 90) # the bin size is set to 10, data  ↩
    range is set to [1, 90]
h.loadData() # update the histogram
g.replot() # update the display

# print histogram values:
for i in range (0, h.dataSize()):
  print i, "Bin start = ", h.x(i), "counts = ", h.y(i)

# print statistic information:
print "Standard deviation = ", h.standardDeviation()
print "Mean = ", h.mean()
```

You can also enable autobinning (a default number of ten bins will be used):

```
 h.setAutoBinning()
```

7.2.10.5 The plot title

```
l.setTitle("My beautiful plot")
l.setTitleFont(QtGui.QFont("Arial", 12))
l.setTitleColor(QtGui.QColor("red"))
l.setTitleAlignment(QtCore.Qt.AlignLeft)
```

The alignment parameter can be any combination of the Qt alignment flags (see the PyQt documentation for more details).

If you want you can remove the plot title using:

```
l.removeTitle()
```

Here's how you can add greek symbols in the plot title or in any other text in the plot layer: axis labels, legends:

```
l.setTitle("normal text <font face=\"Symbol\">greek text</ ↩
    font>")
```

Using the font specifications, you can also change the color of some parts of the title only:

```
l=newGraph().activeLayer()
l.setTitle("<font color = red>red</font> <font color = yellow ↩
    >yellow</font> <font color = blue>blue</font>")
```

7.2.10.6 Customizing the axes

Layer axes can be shown/hidden using the following function:

```
l.enableAxis(int axis, on = True)
```

where `axis` can be any integer value between 0 and 3 or the equivalent reserved word:

0. Layer.Left

1. Layer.Right

2. Layer.Bottom

3. Layer.Top

If an axis is enabled, you can fully customize it via a Python script. For example you can set its title:

```
l.setAxisTitle(axis, "My axis title")
l.setAxisTitleFont(axis, QtGui.QFont("Arial", 11))
l.setAxisTitleColor(axis, QtGui.QColor("blue"))
l.setAxisTitleAlignment(axis, alignFlags)
```

its color and the font used for the tick labels:

```
l.setAxisColor(axis, QtGui.QColor("green"))
l.setAxisFont(axis, QtGui.QFont("Arial", 10))
```

The tick labels of an axis can be enabled or disabled, you can set their color and their rotation angle:

```
l.enableAxisLabels(axis, on = True)
l.setAxisLabelsColor(axis, QtGui.QColor("black"))
l.setAxisLabelRotation(axis, angle)
```

`angle` can be any integer value between -90 and 90 degrees. A rotation angle can be set only for horizontal axes (Bottom and Top).

The numerical format of the labels can be set using:

```
l.setAxisNumericFormat(axis, format, precision = 6, formula)
```

where `format` can have the following values:

0. Automatic: the most compact numeric representation is chosen

1. Decimal: numbers are displayed in floating point form

2. Scientific: numbers are displayed using the exponential notation

3. Superscripts: like Scientific, but the exponential part is displayed as a power of 10

`precision` is the number of significant digits and `formula` is a mathematical expression that can be used to link oposite scales. It's argument must be `x` for horizontal axes and `y` for vertical axes. For example, assuming that the bottom axis displays a range of wavelengths in nanometers and that the top axis represents the equivalent energies in eV, with the help of the code bellow all the wavelengths will be automatically converted to electron-volts and the result will be displayed in floating point form with two significant digits after the decimal dot sign:

```
l.setAxisNumericFormat(Layer.Top, 1, 2, "1239.8419/x")
```

The axis ticks can be customized via the following functions:

```
l.setTicksLength(minLength, majLength)
l.setMajorTicksType(axis, majTicksType)
l.setMinorTicksType(axis, minTicksType)
l.setAxisTicksLength(axis, majTicksType, minTicksType,  ↩
    minLength, majLength)
```

where the `majTicksType` and `minTicksType` parameters specify the desired orientation for the major and minor ticks, respectively:

0. Layer.NoTicks

1. Layer.Out: outward orientation for ticks, with respect to the plot canvas

2. Layer.InOut: both inward and outward ticks

3. Layer.In: inward ticks

`minLength` specifies the length of the minor ticks, in pixels and `majLength` the length of the major ticks.

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 `type` specifies the desired scale type:

0. Layer.Linear

1. Layer.Log10

2. Layer.Ln

3. Layer.Log2

4. Layer.Reciprocal

5. Layer.Probability

6. Layer.Logit

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. Defining a scale range for an axis doesn't automatically disable autoscaling. This means that if a curve is added or removed from the layer, the axes will still automatically adapt to the new data interval. This can be avoided by disabling the autoscaling mode, thus making sure that your scale settings will always be taken into account:

```
l.enableAutoscaling(False)
```

If you want to rescale the plot layer so that all the data points are visible, you can use the following utility function:

```
l.setAutoScale()
```

The same `setScale` function above, with a longer list of arguments, can be used to define an axis break region:

```
l.setScale(axis, start, end, step=0.0, majorTicks=5,  ↩
    minorTicks=5, type=0, inverted=False,
  left=-DBL_MAX, right=DBL_MAX, breakPosition=50,  ↩
      stepBeforeBreak=0.0, stepAfterBreak=0.0,
  minTicksBeforeBreak=4, minTicksAfterBreak=4,  ↩
      log10AfterBreak=False, breakWidth=4, breakDecoration= ↩
      True)
```

where `left` specifies the left limit of the break region, `right` the right limit, `breakPosition` is the position of the break expressed as a percentage of the axis length and `breakWidth` is the width of the break region in pixels. The names of the other parameters should be self-explanatory.

Finally, you can specify the width of all axes and enable/disable the drawing of their backbone line, using:

```
l.setAxesLinewidth(2)
l.drawAxesBackbones(True)
```

7.2.10.7 The canvas

You can display a rectangular frame around the drawing area of the plot (the canvas) and fill it with a background color, using:

```
l.setCanvasFrame(2, QtGui.QColor("red"))
l.setCanvasColor(QtGui.QColor("lightGrey"))
```

Drawing the canvas frame and disabling the axes backbone lines is the only possible solution for the issue of axes not touching themselves at their ends.

7.2.10.8 The layer frame

You can display a rectangular frame around the whole layer and fill it with a background color, using:

```
l.setFrame(2, QtGui.QColor("blue"))
l.setBackgroundColor(QtGui.QColor("grey"))
```

The default spacing between the layer frame and the other layer elements (axes, title) can be changed via:

```
l.setMargin(10)
```

7.2.10.9 Customizing the grid

You can display the grid associated to a layer axis or the whole grid using:

```
l.showGrid(axis)
l.showGrid()
```

This will display the grid with the default color, width and pen style settings. If you need to change these settings, as well as to enable/disable certain grid lines, you can use the following functions:

```
grid = l.grid()
grid.setMajPenX(QtGui.QPen(QtCore.Qt.red, 1))
grid.setMinPenX(QtGui.QPen(QtCore.Qt.yellow, 1, QtCore.Qt. ↩
    DotLine))
grid.setMajPenY(QtGui.QPen(QtCore.Qt.green, 1))
grid.setMinPenY(QtGui.QPen(QtCore.Qt.blue, 1, QtCore.Qt. ↩
    DashDotLine))
grid.enableXMax(True)
grid.enableXMin()
grid.enableYMax()
grid.enableYMin(False)
grid.enableZeroLineX(True)
grid.enableZeroLineY(False)
grid.setXZeroLinePen(QtGui.QPen(QtCore.Qt.black, 2))
grid.setYZeroLinePen(QtGui.QPen(QtCore.Qt.black, 2))
l.replot()
```

All the grid functions containing an `X` refer to the vertical grid lineas, whereas the `Y` leter indicates the horizontal ones. Also, the `Maj` word refers to the main grid lines and `Min` to the secondary grid.

7.2.10.10 The plot legend

You can add a new legend to a plot using:

```
legend = l.newLegend()
#or
legend = l.newLegend("enter your text here")
```

Plot legends are special text objects which are updated each time you add or remove a curve from the layer. They have a special `auto-update` flag which is enabled by default. The following function returns `True` for a legend object:

```
legend.isAutoUpdateEnabled()
```

You can disable/enable the auto-update behavior of a legend/text object using:

```
legend.setAutoUpdate(False/True)
```

You can add common texts like this:

```
text = l.addText(legend)
text.setOrigin(legend.x(), legend.y()+50)
```

Please notice that the `addText` function returns a different reference to the new text object. You can use this new reference later on in order to remove the text:

```
l.remove(text)
```

Once you have created a legend/text, it's very easy to customize it. If you want to modify the text you can use:

```
legend.setText("Enter your text here")
```

All other properties of the legend: rotation angle, text color, background color, frame style, font and position of the top-left corner can be modified via the following functions:

```
legend.setAngle(90)
legend.setTextColor(QtGui.QColor("red"))
legend.setBackgroundColor(QtGui.QColor("yellow"))
legend.setFrameStyle(Frame.Shadow)
legend.setFrameColor(QtCore.Qt.red)
legend.setFrameWidth(3)
legend.setFrameLineStyle(QtCore.Qt.DotLine)
legend.setFont(QtGui.QFont("Arial", 14, QtGui.QFont.Bold,  ↩
    True))
# set top-left position using scale coordinates:
legend.setOriginCoord(200.5, 600.32)
```

```
# or set top-left position using pixel coordinates:
legend.setOrigin(5, 10)
legend.repaint()
```

Other frame styles available for legends are: `Legend.Line`, which draws a rectangle around the text and `Legend.None` (no frame at all). There is also a function allowing you to add an automatically built time stamp:

```
timeStamp = l.addTimeStamp()
```

7.2.10.11 Adding arrows/lines to a plot layer

```
arrow = ArrowMarker()
arrow.setStart(10.5, 12.5)
arrow.setEnd(200, 400.51)
arrow.setStyle(QtCore.Qt.DashLine)
arrow.setColor(QtGui.QColor("red"))
arrow.setWidth(1)
arrow.drawStartArrow(False)
arrow.drawEndArrow(True)
arrow.setHeadLength(7)
arrow.setHeadAngle(35)
arrow.fillArrowHead(True)

l = newGraph().activeLayer()
arrow1 = l.addArrow(arrow)

arrow.setStart(120.5, 320.5)
arrow.setColor(QtGui.QColor("blue"))
arrow2 = l.addArrow(arrow)

l.remove(arrow1)
```

As you might notice from the sample code above, the `addArrow` function returns a reference to a new arrow object that can be used later on to modify this new arrow or to delete it with the `remove` function.

It is possible to modify the properties of all the lines/arrows in a plot layer, see the short example bellow:

```
g = graph("Graph1").activeLayer()
lst = g.arrowsList()
for i in range (0, g.numArrows()):
  lst[i].setColor(Qt.green)

g.replot()
```

7.2.10.12 Adding images to a layer

```
l = newGraph().activeLayer()
image = l.addImage("C:/poze/adi/PIC00074.jpg")
image.setCoordinates(200, 800, 800, 200)
image.setFrameStyle(Frame.Shadow)
image.setFrameColor(QtCore.Qt.green)
image.setFrameWidth(3)
l.replot()
```

The `setCoordinates` function above can be used to set the geometry of the image using scale coordinates. If you need to specify the image geometry in pixel coordinates, independently of the plot axes values, you may use the following functions:

```
image.setOrigin(x, y)
image.setSize(width, height)
image.setRect(x, y, width, height)
l.replot()
```

You can remove an image using its reference:

```
l.remove(image)
```

7.2.10.13 Rectangles

```
l = newGraph().activeLayer()

r = Rectangle(l)
r.setSize(100, 100)
r.setOrigin(100, 200)
r.setBackgroundColor(QtCore.Qt.yellow)
r.setFrameColor(QtCore.Qt.red)
r.setFrameWidth(3)
r.setFrameLineStyle(QtCore.Qt.DotLine)
r.setBrush(QtGui.QBrush(QtCore.Qt.green, QtCore.Qt. ↩
    FDiagPattern))

r1 = l.add(r)
```

You can remove a rectangle using its reference:

```
r2 = l.add(r)
r2.setOrigin(200, 200)
l.remove(r1)
```

7.2.10.14 Circles/Ellipses

```
l = newGraph().activeLayer()

e = Ellipse(l)
e.setSize(100, 100)
e.setOrigin(100, 200)
e.setBackgroundColor(QtCore.Qt.yellow)
e.setFrameColor(QtCore.Qt.red)
e.setFrameWidth(0.8)
e.setFrameLineStyle(QtCore.Qt.DotLine)
e.setBrush(QtGui.QBrush(QtCore.Qt.green, QtCore.Qt. ↩
    FDiagPattern))

l.add(e)
```

7.2.10.15 Antialiasing

Antialiasing can be enabled/disabled for the drawing of the curves and other layer objects, but it is a very resources consuming feature:

```
l.setAntialiasing(True, bool update = True)
```

7.2.10.16 Resizing layers

A layer can be resized using the methods bellow, where the first argument is the new width, the second is the new height and sizes are defined in pixels:

```
 l.resize(200, 200);
 l.resize(QSize(w, h))
```

If you also need to reposition the layer, you can use the following functions, where the first two arguments specify the new position of the top left corner of the canvas:

```
 l.setGeometry(100, 100, 200, 200);
 l.setGeometry(QRect(x, y, w, h));
```

The default behaviour of 2D plot layers, with respect to the resizing of the graph window is to adapt the sizes of the fonts used for the various texts, to the new size of the plot window. You can override this behaviour and keep the size of the fonts unchanged:

```
l.setAutoscaleFonts(False)
```

7.2.10.17 Resizing the drawing area

The drawing area of a layer (the canvas) can be resized using the methods bellow, where the first argument is the new width, the second is the new height and sizes are defined in pixels:

```
l.setCanvasSize(200, 200);
l.setCanvasSize(QSize(w, h))
```

If you also need to reposition the canvas, you can use the following functions, where the first two arguments specify the new position of the top left corner of the canvas:

```
l.setCanvasGeometry(100, 100, 200, 200);
l.setCanvasGeometry(QRect(x, y, w, h));
```

Please keep in mind that the fonts of the layer are not rescaled when you resize the layer canvas using the above methods.

7.2.10.18 Exporting plots/layers to different image formats

Layers and whole Graphs can be printed and exported from within Python. The fastest way to export a plot/layer is the following:

```
l.export(fileName)
```

This function uses some default parameters for the properties of the image. If you need more control over the exported images you can use one of the following specialized functions:

```
l.exportVector(fileName, dpi = 96, color = True, size =  ↩
    QSizeF(), unit = Frame.Pixel, fontsFactor = 1.0)
l.exportImage(fileName, quality = 100, transparent = False,  ↩
    dpi = 0, size = QSizeF(), unit = Frame.Pixel, fontsFactor ↩
     = 1.0)
l.exportTex(fileName, color = True, escapeStrings = True,  ↩
    fontSizes = True, size = QSizeF(), unit = Frame.Pixel,  ↩
    fontsFactor = 1.0)
```

The function `exportVector` can export the plot/layer to the following vector formats: .eps, .ps, .pdf.

The function `exportImage` can be used if you need to export to one of the Qt supported bitmap image formats (.bmp, .png, .jpg, etc...). The `transparent` option can only be used in conjunction with the file formats supporting transprency: .png and .tif (.tiff). The `quality` parameter influences the size of the output file. The higher this value (maximum is 100), the higher the qualitity of the image, but the larger the size of the resulting files. The `dpi` parameter represents the export resolution in pixels per inch (the default is screen resolution), `size` is the printed size of the image (the default is the size on screen) and `unit` is the length unit used to express the custom size and can take one of the following values:

0. Inch

1. Millimeter

2. Centimeter

3. Point: 1/72th of an inch

4. Pixel

The `fontsFactor` parameter represents a scaling factor for the font sizes of all texts in the plot (the default is 1.0, meaning no scaling). If you set this parameter to 0, the program will automatically try to calculate a scale factor.

The function `exportTex` can be used if you need to export to a TeX file. The `escapeStrings` parameter enables/disables the escaping of special TeX characters like: $, {, }, ^, etc... If `True`, the `fontSizes` parameter triggers the export of the original font sizes in the plot layer. Otherwise all exported text strings will use the font size specified in the preamble of the TeX document.

All the export functions rely on the file name suffix in order to choose the image format.

7.2.11 Arranging Layers

When you are working with many layers in a 2D plot window, setting the layout of these layers manually can be a very tedious task. With the help of a simple Python script you can make this task very easy and automatically manage the layout of the plot window. For example, here's how you can create a two rows by two columns matrix of layers, each plot layer having a canvas size (the drawing area) of 400 pixels wide and 300 pixels in height:

```
g = newGraph("Test", 4, 2, 2)
g.setLayerCanvasSize(400, 300)
g.arrangeLayers(False, True)
```

The `arrangeLayers()` function takes two parameters. The first one specifies if the layers should be arranged automatically, using a best-layout algorithm, or if the numbers of rows and columns is fixed by the user. If the value of the second parameter is `True`, the size of the canvas is fixed by the user and the plot window will be enlarged or shrinked, according to the user settings. Otherwise the size of the plot window will be kept and the canvas area of each layer will be automatically adapted to fit this size. Here's how you can modify the graph created in the previous example, in order to display a row of three layers, while keeping the size of the plot window unchanged:

```
g.setNumLayers(3)
g.setRows(1)
g.setCols(3)
g.arrangeLayers(False, False)
```

By default, the space betwee two neighbouring layers as well as the distance between the layers and the borders of the plot window is set to five pixels. You can change the spacing between the layers and the margins using the following functions:

```
g.setSpacing (x, y)
g.setMargins (left, right, top, bottom)
```

Another aspect of the layout management is the alignement of the layers. There are three alignement flags that you can use for the horizontal alignement (HCenter, Left, Right) and another three for the vertical alignement (VCenter, Top, Bottom) of the layers. The following code line aligns the layers with the right edge of the window and centers them vertically in the available space:

```
g.setAlignement(Graph.Right, Graph.VCenter)
```

All the examples above suppose that the layers are aranged on a grid, but of course you can add layers at any position in the plot window. In the examples bellow the x, y coordinates, in pixels, refer to the position of the top-left corner of the layer. The origin of the coordinates system coincides with the top-left corner of the plot window, the y coordinate increasing towards the bottom of the window. If the width and height of the layer are not specified they will be set to the default values. The last argument specifies if the default preferences, specified via the Preferences dialog, will be used to customize the new layer (default value is `False`):

```
g = newGraph()
l1 = g.addLayer()
l2 = g.addLayer(215, 20)
l3 = g.addLayer(10, 20, 200, 200)
l4 = g.addLayer(10, 20, 200, 200, True)
```

You can remove a plot layer using:

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

As you have already seen, in a plot window the active layer is, by default, the last layer added to the plot, but you can change it programatically:

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

In case you need to perform a repetitive task on all the layers in a plot window, you need to use a for loop and of course you need to know the number of layers existant on the plot. Here's a small example showing how to custom the titles of all the layers in the plot window:

```
g = graph("Graph1")
layers = g.numLayers()
for i in range (1, layers+1):
  l = g.layer(i)
  l.setTitle("Layer"+QtCore.QString.number(i))
  l.setTitleColor(QtGui.QColor("red"))
  l.setTitleFont(QtGui.QFont("Arial", 14, QtGui.QFont.Bold,  ↩
      True))
  l.setTitleAlignment(QtCore.Qt.AlignLeft)
```

Finally, sometimes it might be useful to be able to swap two layers. This can be done with the help of the following function:

```
g.swapLayers(layerNum1, layerNum2)
```

7.2.12 Waterfall Plots

Waterfall plots use a cascading layout for the layers in a 2D plot window. You can create and customize them using the functions bellow:

```
g = waterfallPlot(table("Table1"), (2, 3, 4))
g.setWaterfallOffset(15, 10)
g.setWaterfallSideLines(True) # draw side lines for all the curves displayed
g.setWaterfallFillColor(Qt.lightGray)
g.reverseWaterfallOrder() # reverse the order of the displayed curves
```

7.2.13 3D Plots

7.2.13.1 Creating a 3D plot

You can plot 3D analytical functions or parametric surfaces. For the 3D functions, the only parameters allowed are `x` for the the abscissae values and `y` for the ordinates:

```
g = plot3D("sin(x*y)", -10.0, 10.0, -10.0, 10.0, -2.0, 2.0)
```

For the parametric surfaces the only parameters allowed are the latitude and the longitude: `u` and `v`. Here's, for example, how you can plot a sphere:

```
g = plot3D("cos(u)*cos(v)", "sin(u)*cos(v)", "sin(v)", -3.14, 3.14, -2, 2)
```

You can also create 3D height maps using data from matrices and, of course, you can plot table columns:

```
g = plot3D(matrix("Matrix1"), style = 5)
g = plot3D(table("Table1"), "3", style)
```

In the case of 3D plots created from matrix data sources the `style` parameter can take any integer value from 1 to 5, with the following signification:

1. Wireframe style

2. Hidden Line style

3. Color filled polygons without edges

4. Color filled polygons with separately colored edges

5. Scattered points (the default style)

For 3D plots created from tables the `style` parameter can take any integer value from 0 to 3 or the equivalent style values from the follwing list:

0. Graph3D.Scatter

1. Graph3D.Trajectory

2. Graph3D.Bars

3. Graph3D.Ribbon

An alternative method to create a 3D plot is to create an empty plot window and to assign a data source to it. As you have already seen a data source can be an analytical function, a matrix or a table. For large data sets you can increase the drawing speed by reducing the number of points taken into account. The lower the resolution parameter, the higher the number of points used: for an integer value of 1, all the data points are drawn.

```
g = newPlot3D("test3D")
g.setTitle("My 3D Plot", QtGui.QColor("blue"), QtGui.QFont(" ↩
    Arial",14))
g.setResolution(2)
g.setFunction("sin(x*y)", -10.0, 10.0, -10.0, 10.0, -2.0,  ↩
    2.0)
#or
g.setData(table("Table1"), "3")
#or
g.setMatrix(matrix("Matrix1"))
```

Once a plot is created, you can modify the scales and set the data range to display, using, for example:

```
g.setScales(-1.0, 1.0, -10.0, 11.0, -2.0, 3.0)
```

7.2.13.2 Customizing the view

When a new 3D plot is created, the scene view parameters are set to default values. Of course, QtiPlot provides functions to customize each aspect of the view. For example, you can set rotation angles, in degrees, around the X, Y and Z axes, respectively, using:

```
g.setRotation(45, 15, 35)
```

The following function allows you to shift the plot along the world X, Y and Z axes, respectively:

```
g.setShift(3.0, 7.0, -4.0)
```

You can also zoom in/out the entire plot as a whole, or you can zoom along a particular axis:

```
g.setZoom(10)
g.setScale(0.1, 0.05, 0.3)
```

Also, you can automatically detect the zoom values that fit best with the size of the plot window:

```
g.findBestLayout()
```

You can enable/disable the perspective view mode or animate the view using:

```
g.setOrthogonal(False)
g.animate(True)
```

7.2.13.3 Plot Styles

The style of the 3D plot can be set using the following functions:

```
g.setPolygonStyle()
g.setFilledMeshStyle()
g.showLegend(True)
g.setHiddenLineStyle()
g.setWireframeStyle()
g.setAntialiasing(True)
g.setMeshLineWidth(0.7)
```

For scatter plots using points you can specify the radius of the points and their shape: circles if `smooth` is True, rectangles otherwise.

```
g.setDotOptions(10, smooth = True)
g.setDotStyle()
```

Other symbols available for scatter plots are: bars

```
g.setBarRadius(0.01)
g.setBarLines(False)
g.setFilledBars(True)
g.setBarStyle()
```

cones

```
g.setConeOptions(radius, quality)
g.setConeStyle()
```

and crosses (surrounded by a box frame, if `boxed` is set to True):

```
g.setCrossOptions(radius, width, smooth, boxed)
g.setCrossStyle()
```

7.2.13.4 The 2D Projection

By default the floor projection of the 3D surface plot is disabled. You can enable a full 2D projection or only display the isolines using the following functions:

```
g.showFloorProjection()
g.showFloorIsolines()
g.setEmptyFloor()
```

7.2.13.5 Customizing the Coordinates System

The coordinates system around the surface plot can be customized to display all the twelve axes, only three of them or none, respectively, with the help of the following funcions:

```
g.setBoxed()
g.setFramed()
g.setNoAxes()
```

If the axes are enabled, you can set their legends and the distance between the legend and the axes via:

```
g.setXAxisLabel("X axis legend")
g.setYAxisLabel("Y axis legend")
g.setZAxisLabel("Z axis legend")
g.setLabelsDistance(30)
```

It is possible to set the numerical format and precision of the axes using the function bellow:

```
g.setAxisNumericFormat(axis, format, precision)
```

where the first parameter is the index of the axis: 0 for X, 1 for Y and 2 for Z, the second one is the numerical format:

0. Graph3D.Default: decimal or scientific, depending which is most compact

1. Graph3D.Decimal: 10000.0

2. Graph3D.Scientific: 1e4

3. Graph3D.Engineering: 10k

and the last parameter is the precision (the number of significant digits). The following convenience functions are also provided, where you don't have to specify the index of the axis anymore:

```
g.setXAxisNumericFormat(1, 3)
g.setYAxisNumericFormat(1, 3)
g.setZAxisNumericFormat(1, 3)
```

Also, you can fix the length of the major and minor ticks of an axis:

```
g.setXAxisTickLength(2.5, 1.5)
g.setYAxisTickLength(2.5, 1.5)
g.setZAxisTickLength(2.5, 1.5)
```

7.2.13.6 Grid

If the coordinate system is displayed, you can also display a grid around the surface plot. Each side of the grid can be shown/hidden:

```
g.setLeftGrid(True)
g.setRightGrid()
g.setCeilGrid()
g.setFloorGrid()
g.setFrontGrid()
g.setBackGrid(False)
```

7.2.13.7 Customizing the Plot Colors

The default color map of the plot is defined using two colors: red for the maximum data values and blue for the minimum data values. You can change these default colors:

```
g.setDataColors(QtCore.Qt.black, QtCore.Qt.green)
g.update()
```

Of course, you can define more complex color maps, using *LinearColorMap* objects:

```
map = LinearColorMap(QtCore.Qt.yellow, QtCore.Qt.blue)
map.setMode(LinearColorMap.FixedColors) # default mode is  ↩
    LinearColorMap.ScaledColors
map.addColorStop(0.2, QtCore.Qt.magenta)
map.addColorStop(0.7, QtCore.Qt.cyan)
g.setDataColorMap(map)
g.update()
```

Also, you can use predifined color maps stored in .map files. A .map file consists of a of 255 lines, each line defining a color coded as RGB values. A set of predefined color map files can be downloaded from QtiPlot web site, in the "Miscelanous" section.

```
g.setDataColorMap(fileName)
g.update()
```

The colors of all the other plot elements can be customized as shown bellow. Don't forget to update the plot in order to display the new colors:

```
g.setMeshColor(QtGui.QColor("blue"))
g.setAxesColor(QtGui.QColor("green"))
g.setNumbersColor(QtGui.QColor("black"))
g.setLabelsColor(QtGui.QColor("darkRed"))
g.setBackgroundColor(QtGui.QColor("lightYellow"))
g.setGridColor(QtGui.QColor("grey"))
g.setDataColors(QtGui.QColor("red"), QtGui.QColor("orange"))
g.setOpacity(0.75)
g.update()
```

7.2.13.8 Exporting

In order to export a 3D plot you need to specify a file name containing a valid file format extention:

```
g.export(fileName)
```

This function uses some default export options. If you want to customize the exported image, you should use the following function in order to export to raster image formats:

```
g.exportImage(fileName, int quality = 100, bool transparent = ↩
     False, dpi = 0, size = QSizeF(), unit = Frame.Pixel,  ↩
    fontsFactor = 1.0)
```

where `quality` is a compression factor: the larger its value, the better the quality of the exported image, but also the larger the file size. The `dpi` parameter represents the export resolution in pixels per inch (the default is screen resolution), `size` is the printed size of the image (the default is the size on screen) and `unit` is the length unit used to express the custom size and can take one of the following values:

0. Inch

1. Millimeter

2. Centimeter

3. Point: 1/72th of an inch

4. Pixel

The `fontsFactor` parameter represents a scaling factor for the font sizes of all texts in the plot (the default is 1.0, meaning no scaling). If you set this parameter to 0, the program will automatically try to calculate a scale factor.

3D plots can be exported to any of the following vector formats: .eps, .ps, .pdf, .pgf and .svg, using the function bellow:

```
g.exportVector(fileName, textMode = 0, sortMode = 1, size =  ↩
    QSizeF(), unit = Frame.Pixel, fontsFactor = 1.0)
```

where `textMode` is an integer value, specifing how texts are handled. It can take one of the following values:

0. All text will be converted to bitmap images (default).

1. Text output in the native output format.

2. Text output in additional LaTeX file as an overlay.

The `sortMode` parameter is also an integer value and can take one of the following values:

0. No sorting at all.

1. A simple, quick sort algorithm (default).

2. BSP sort: best algorithm, but slow.

The other parameters have the same meaning as for the export of 2D plots.

7.2.14 Data Analysis

7.2.14.1 General Functions

As you will see in the following subsections, the data analysis operations available in QtiPlot are: convolution/deconvolution, correlation, differentiation, FFT, filtering, smoothing, fitting and numerical integration of data sets. Generally, you can declare/initialize an analysis operation using one of the following methods, depending on the data source, which can be a 2D plot curve or a table:

```
op = LogisticFit(graph("Graph1").activeLayer().curve(0),  ↩
    15.2, 30.9)
op = FFTFilter(graph("Graph1").activeLayer(), "Table1_2",  ↩
    1.5, 3.9)
op = LinearFit(table("Table1"), "colX", "colY", 10, 100)
```

In the first example the data source is a curve *Table1_2*, plotted in the active layer of the graph *Graph1* and the abscissae range is chosen between 1.5 and 3.9. In the second example the data source is a table *Table1*. The abscissae of the data set are stored in the column called *colX* and the ordinates in the column *colY*. The data range is chosen between the 10th row and the row with the index 100. If you don't specify the row range, by default the whole table will be used. Not all operations support curves as data sources, like for example: convolution/deconvolution and correlation. For these operations only table columns can be used as data sources for the moment.

Once you have initialized an operation, you can still chage its input data via the following functions:

```
op.setDataFromCurve(graph("Graph2").activeLayer().curve(1),  ↩
    10.5, 20.1)
op.setDataFromCurve("Table1_energy", 10.5, 20.1, graph(" ↩
    Graph2").activeLayer())
op.setDataFromTable(table("Table1"), "colX", "colY", 10, 100)
```

You don't have to specify a plot layer in the setDataFromCurve() function, if the analysis operation has already been initialized by specifying a curve on an existing graph and you just want to treat another curve from the same plot layer.

Also, when performing analysis tasks via Python scripts, there are several utility functions that can be called for all operations. For example you can disable any graphical output from an operation or you can redirect the output to the plot layer of your choice:

```
op.enableGraphicsDisplay(False)
op.enableGraphicsDisplay(True, graph("Graph2").activeLayer())
```

Let's assume that you need to perform a specific operation `op`, which analyses your data and at the end, displays a result curve. For this kind of operations, you can customize the number of points in the resulting curve and its color:

```
op.setOutputPoints(int)
op.setColor(int)
op.setColor("green")
```

Colors can be specified by their names or as integer values, from 0 to 23, each integer corresponding to a predefined color: 0 - "black", 1 - "red", 2 - "green", 3 - "blue", 4 - "cyan", 5 - "magenta", 6 - "yellow", 7 - "darkYellow", 8 - "navy", 9 - "purple", etc ...

Most of the time, a new table is also created as a result of a data analysis operation. This table stores the data displayed by the result curve and is hidden by default, but you can interact with it via the following function:

```
t = op.resultTable()
```

After the initialization of an analysis operation, which consists of setting the data source, the data range and some other properties, like color, number of points, etc..., you can execute it via a call to its run() function:

```
op.run()
```

For data fitting operations, there's an alias for the run() function which is: fit().

7.2.14.2 Correlation, Convolution/Deconvolution

Assuming you have a table named "Table1", here's how you can calculate the convolution of two of its columns, "Table1_B" and "Table1_C":

```
conv = Convolution(table("Table1"),  "B", "C")
conv.setColor("green")
conv.run()
```

The deconvolution and the correlation of two data sets can be done using a similar synthax:

```
dec = Deconvolution(table("Table1"),  "B", "C")
dec.run()

cor = Correlation(table("Table1"),  "B", "C", 10, 200)
cor.setColor("green")
cor.run()
```

7.2.14.3 Differentiation

Assuming you have a Graph named "Graph1" containing one curve (on its active layer), here's how you can differentiate this curve within a defined x interval, [2,10] in this case:

```
diff = Differentiation(graph("Graph1").activeLayer().curve(0) ↩
    , 2, 10)
diff.run()
```

The result of these code sequence would be a new plot window displaying the derivative of the initial curve. The numerical derivative is calculated using a five terms formula.

7.2.14.4 FFT

Assuming you have a graph named "Graph1" containing one curve on its active layer and having a periodicity of 0.1 in the time domain, a FFT will allow you to extract its characteristic frequencies. The results will be stored in a hidden table named "FFT1".

```
fft = FFT(graph("Graph1").activeLayer().curve(0))
fft.normalizeAmplitudes(False)
fft.shiftFrequencies(False)
fft.setSampling(0.1)
fft.run()
```

By default the calculated amplitudes are normalized and the corresponding frequencies are shifted in order to obtain a centered x-scale. If we want to recover the initial curve with the help of the inverse transformation, we mustn't modify the amplitudes and the frequencies. Also the sampling parameter must be set to the inverse of the time period, that is 10. Here's how we can perform the inverse FFT, using the "FFT1" table, in order to recover the original curve:

```
ifft = FFT(table("FFT1"), "Real", "Imaginary")
ifft.setInverseFFT()
ifft.normalizeAmplitudes(False)
ifft.shiftFrequencies(False)
ifft.setSampling(10)
ifft.run()
```

7.2.14.5 FFT Filters

In this section, it will be assumed that you have a signal displayed in a graph ("Graph1", on its active layer). This signal has a power spectrum with high and low frequencies. You can filter some of these frequencies according to your needs, using a FFTFilter. Here's how you can cut all the frequencies lower than 1 Hz:

```
filter = FFTFilter(graph("Graph1").activeLayer().curve(0),  ↩
    FFTFilter.HighPass)
filter.setCutoff(1)
filter.run()
```

Here's how you can cut all the frequencies lower than 1.5 Hz and higher than 3.5 Hz. In the following example the continuous component of the signal is also removed:

```
filter.setFilterType(FFTFilter.BandPass)
filter.enableOffset(False)
filter.setBand(1.5, 3.5)
filter.run()
```

Other types of FFT filters available in QtiPlot are: low pass (`FFTFilter.LowPass`) and band block (`FFTFilter.BandBlock`).

7.2.14.6 Fitting

Assuming you have a graph named "Graph1" displaying a curve entitled "Table1_2" on its active layer, a minimal Fit example would be:

```
f = GaussFit(graph("Graph1").activeLayer().curve(0))
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(curve)
- PolynomialFit(curve, degree=2, legend=False)
- ExponentialFit(curve, growth=False)
- TwoExpFit(curve)
- ThreeExpFit(curve)
- GaussFit(curve)
- GaussAmpFit(curve)
- LorentzFit(curve)
- LogisticFit(curve)
- SigmoidalFit(curve)
- NonLinearFit(curve)

  ```
  f = NonLinearFit(layer, curve)
  f.setFormula(formula_string)
  f.save(fileName)
  ```

- PluginFit(curve)

  ```
  f = PluginFit(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().curve(0), 2, 7)
f.fit()
```

You can also restrict the search range for any of the fit parameters:

```
f = NonLinearFit(graph("Graph1").activeLayer().curve(0))
f.setFormula("a0+a1*x+a2*x*x")
f.setParameterRange(parameterIndex, start, end)
```

All the settings of a non-linear fit can be saved to an XML file and restored later one, using this file, for a faster editing process. Here's for example how you can save the above fit function:

```
f.save("/fit_models/poly_fit.txt")
```

and how you can use this file during another fitting session, later on:

```
f = NonLinearFit(graph("Graph1").activeLayer(), "Table1_2")
f.load("/fit_models/poly_fit.txt")
f.fit()
```

If your script relies on a specific numbering of the fit parameters use setParameters() before setting the formula and switch of automatic detection of the fit parameters when the fit formula is set:

```
f.setParameters("a2","a0","a1")
f.setFormula("a0+a1*x+a2*x*x",0)
```

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

```
f.setDataFromTable(table("Table4"), "w", "energy", 10, 200)  ↩
    change data source
f.setDataFromCurve(curve)     change data source
f.setDataFromCurve(curveTitle, graph)   change data source
f.setDataFromCurve(curve, from, to)   change data source
f.setDataFromCurve(curveTitle, 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.setWeightingData(method, colname) # method = Fit. ↩
    NoWeighting, Fit.Instrumental, Fit.Statistical, Fit. ↩
    Dataset, Fit.Direct
f.setTolerance(tolerance)
f.setOutputPrecision(precision)
```

```
f.setMaximumIterations(number)
f.scaleErrors(yes = True)
f.setColor("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.residuals()
f.dataSize()
f.numParameters()
f.parametersTable("params")
f.covarianceMatrix("cov")
```

There are a number of statistical functions allowing you to test the goodness of the fit:

```
f.chiSquare()
f.rSquare()
f.adjustedRSquare()
f.rmse() # Root Mean Squared Error
f.rss()  # Residual Sum of Squares
```

Also you can display the confidence and the prediction limits for the fit, using a custom confidence level:

```
f.showPredictionLimits(0.95)
f.showConfidenceLimits(0.95)
```

Confidence limits for individual fit parameters can be calculated using:

```
f.lcl(parameterIndex, confidenceLevel)
f.ucl(parameterIndex, confidenceLevel)
```

where `parameterIndex` is a value between zero and f.numParameters() - 1.

It is important to know that QtiPlot can generate an analytical formula for the resulting fit curve or a normal plot curve with data stored in a hidden table. You can choose either of these two output options, before calling the fit() instruction, using:

```
f.generateFunction(True, points=100)
```

If the first parameter of the above function is set to True, QtiPlot will generate an analytical function curve. If the `points` parameter is not specified, by default the function will be estimated over 100 points. You can get the analytical formula of the fit curve via a call to resultFormula():

```
formula = f.resultFormula()
print(formula)
```

If the first parameter of generateFunction() is set to False, QtiPlot will create a hidden data table contining the same number of points as the data set/curve to be fitted (same

abscissae). You can interact with this table and extract the data points of the result fit curve using:

```
t = f.resultTable()
```

7.2.14.7 Integration

With the same assumptions as above, here's how you can integrate a curve within a given interval:

```
integral = Integration(graph("Graph1").activeLayer().curve(0) ↩
    , 2, 10)
integral.setMethodOrder(4)
integral.setTolerance(1e-4)
integral.setMaximumIterations(100)
integral.run()
result = integral.area()
```

The method order parameter can be any integer value between 1 (Trapezoidal rule, the default value) and 5. The code integrates the curve using an iterative algorithm. The tolerance determines the termination criteria for the solver. Because, sometimes we ask for too much accuracy, setting a maximum number of iterations makes sure that the solver will not enter an infinite loop, which could freeze the application.

As you can see from the above example, the numerical value of the integral can be obtained via the `area()` function.

7.2.14.8 Interpolation

The interpolation is used to generate a new data curve with a high number of points from an existing data set. Here's an example:

```
interpolation = Interpolation(graph("Graph1").activeLayer(). ↩
    curve(0), 2, 10, Interpolation.Linear)
interpolation.setOutputPoints(10000)
interpolation.setColor("green")
interpolation.run()
```

The simplest interpolation method is the linear method. There are two other methods available: `Interpolation.Akima` and `Interpolation.Cubic`. You can choose the interpolation method using:

```
interpolation.setMethod(Interpolation.Akima)
```

7.2.14.9 Smoothing

Assuming you have a graph named "Graph1" with an irregular curve entitled "Table1_2" (on its active layer). You can smooth this curve using a SmoothFilter:

```
smooth = SmoothFilter(graph("Graph1").activeLayer().curve(0), ↩
    SmoothFilter.Average)
smooth.setSmoothPoints(10)
smooth.run()
```

The default smoothing method is the mowing window average. Other smoothing methods are the `SmoothFilter.FFT`, `SmoothFilter.Lowess` and `SmoothFilter.SavitzkyGolay`. Here's an example of how to use the last two methods:

```
smooth.setMethod(SmoothFilter.Lowess)
smooth.setLowessParameter(0.2, 2)
smooth.run()
```

```
smooth.setSmoothPoints(5,5)
smooth.setMethod(SmoothFilter.SavitzkyGolay)
smooth.setPolynomOrder(9)
smooth.run()
```

7.2.15 Working with Notes

The following functions are available when dealing with multi-tab notes:

```
setAutoexec(on = True)

text()
setText(text)

exportPDF(fileName)
saveAs(fileName)
importASCII(fileName)

showLineNumbers(on = True)

setFont(QFont f)
setTabStopWidth(int length)

tabs()
addTab()
removeTab(tabIndex)
renameTab(tabIndex, title)

e = editor(int index)
e = currentEditor()
```

7.2.16 Using Qt's dialogs and classes

Let's assume that you have a lot of ASCII data files to analyze. Furthermore, let's suppose that these files were created during several series of measurements, each mea-

surement generating a set of files identified by a certain string in the file name, like for example: "disper1". In order to analyze these files, you need first of all to import them into tables. The following code snippet shows how to automize this task using Qt dialogs and convenience classes:

```
# Pop-up a file dialog allowing to chose the working folder:
dirPath = QtGui.QFileDialog.getExistingDirectory(qti.app, " ↩
    Choose Working Folder", "/test/")

# Create a folder object using Qt's QDir class:
folder = QtCore.QDir(dirPath)

# Pop-up a text input dialog allowing to chose the file  ↩
    naming pattern:
namePattern = QtGui.QInputDialog.getText(qti.app, "Enter  ↩
    Pattern", "Text: ", QtGui.QLineEdit.Normal, "disper1")

# Get the list of file names in the working directory  ↩
    containing the above pattern:
fileNames = folder.entryList (QtCore.QStringList ("*_" +  ↩
    namePattern[0] + "*.dat"))

# Import each file into a new project table:
for i in range (0, lst.count()):
    t = newTable()
    t.importASCII(dirPath + fileNames[i], " ", 0, False, True ↩
        , True)
```

For a detailed description of all the dialogs and utility classes provided by Qt/PyQt please take a look at the PyQt documentation.

7.2.17 Using Qt Designer for easy creation of custom user dialogs

Writing and designing user dialogs can be a very complicated task. The QtDesigner application provided by the Qt framework makes it easy and very pleasant. It allows you to design widgets, dialogs or complete main windows using on-screen forms and a simple drag-and-drop interface. Qt Designer uses XML .ui files to store designs. Once you have finished the design process you can load and use an .ui file in your Python scripts with the help of the `uic` module.

As an example, suppose that we have created a test dialog containing an input QDoubleSpinBox called "valueBox" and a QPushButton called "okButton". On pressing this button, we would like to create a new table displaying the input value in its first cell. We have saved this dialog to a file called "myDialog.ui". A minimalistic approach is shown in the small script bellow:

```
from PyQt4 import uic

global ui, createTable
```

```
def createTable():
  t = newTable()
  t.setCell(1, 1, ui.valueBox.value())

ui = uic.loadUi("myDialog.ui")
ui.connect(ui.okButton, QtCore.SIGNAL("clicked()"),   ↩
    createTable)
ui.show()
```

For more details about how to use .ui files in your Python scripts please read the PyQt4 documentation.

7.2.18 Task automatization example

Bellow you can find a detailed example showing how to completely automatize tasks in QtiPlot. It can be used in order to verify the accuracy of the curve fitting algorithms in QtiPlot. The data used in this example is retrieved from the Statistical Reference Datasets Project of the National Institute of Standards and Technology (NIST). In order to run this example, you need an internet connection, since the script will try to download all the nonlinear regression test files from the Statistical Reference Datasets Project.

```
import urllib, re, sys

# Pop-up a file dialog allowing to chose a destination folder ↩
    :
dirPath = QtGui.QFileDialog.getExistingDirectory(qti.app, " ↩
    Choose Destination Folder")

saveout = sys.stdout
# create a log file in the destination folder
fsock = open(dirPath + "/" + "results.txt", "w")
sys.stdout = fsock

# on Unix systems you can redirect the output directly to a  ↩
    console by uncommenting the line bellow:
#sys.stdout = sys.__stdout__

# make sure that the decimal separator is the dot character
qti.app.setLocale(QtCore.QLocale.c())

host = "http://www.itl.nist.gov/div898/strd/nls/data/LINKS/ ↩
    DATA/"
url = urllib.urlopen(host)
url_string = url.read()
p = re.compile( '\w{,}.dat">' )
iterator = p.finditer( url_string )
for m in iterator:
  name = (m.group()).replace("\">", "")
```

```
if (name == "Nelson.dat"):
  continue

url = host + name
print  "\nRetrieving file: " + url
path = dirPath + "/" + name
urllib.urlretrieve( url, path ) # retrieve .dat file to  ↩
    specified location

file = QtCore.QFile(path)
if file.open(QtCore.QIODevice.ReadOnly):
  ts = QtCore.QTextStream(file)
  name = name.replace(".dat", "")
  changeFolder(addFolder(name)) #create a new folder and  ↩
      move to it
  formula = ""
  parameters = 0
  initValues = list()
  certifiedValues = list()
  standardDevValues = list()
  xLabel = "X"
  yLabel = "Y"

  while (ts.atEnd() == False):
    s = ts.readLine().simplified()

    if (s.contains("(y = ")):
      lst = s.split("=")
      yLabel = lst[1].remove(")")

    if (s.contains("(x = ")):
      lst = s.split("=")
      xLabel = lst[1].remove(")")

    if (s.contains("Model:")):
      s = ts.readLine().simplified()
      lst = s.split(QtCore.QRegExp("\\s"))
      s = lst[0]
      parameters = s.toInt()[0]
      ts.readLine()
      if (name == "Roszman1"):
        ts.readLine()
        formula = ts.readLine().simplified()
      else:
        formula = (ts.readLine() + ts.readLine() + ts. ↩
            readLine()).simplified()
      formula.remove("+ e").remove("y =").replace("[", "(") ↩
          .replace("]", ")")
      formula.replace("**", "^").replace("arctan", "atan")
```

```
    if (s.contains("Starting")):
      ts.readLine()
      ts.readLine()
      for i in range (1, parameters + 1):
        s = ts.readLine().simplified()
        lst = s.split(" = ")
        s = lst[1].simplified()
        lst = s.split(QtCore.QRegExp("\\s"))
        initValues.append(lst[1])
        certifiedValues.append(lst[2])
        standardDevValues.append(lst[3])

    if (s.contains("Data: y")):
      row = 0
      t = newTable(name, 300, 2)
      t.setColName(1, "y")
      t.setColumnRole(1, Table.Y)
      t.setColName(2, "x")
      t.setColumnRole(2, Table.X)
      while (ts.atEnd() == False):
        row = row + 1
        s = ts.readLine().simplified()
        lst = s.split(QtCore.QRegExp("\\s"))
        t.setText(1, row, lst[0])
        t.setText(2, row, lst[1])

      g = plot(t, t.colName(1), Layer.Scatter).activeLayer ↩
          ()
      g.setTitle("Data set: " + name + ".dat")
      g.setAxisTitle(Layer.Bottom, xLabel)
      g.setAxisTitle(Layer.Left, yLabel)

      f = NonLinearFit(g, name + "_" + t.colName(1))
      if (f.setFormula(formula) == False) :
        file.close()
        changeFolder(rootFolder())
        continue

      f.scaleErrors()
      for i in range (0, parameters):
        f.setInitialValue(i, initValues[i].toDouble()[0])
      f.fit()
      g.removeLegend()
      f.showLegend()
      print  "QtiPlot Results:\n" + f.legendInfo()

      print  "\nCertified Values:"
      paramNames = f.parameterNames()
      for i in range (0, parameters):
        print  '%s = %s +/- %s' % (paramNames[i],  ↩
```

```
            certifiedValues[i], standardDevValues[i])

        print  "\nDifference with QtiPlot results:"
        results = f.results()
        for i in range (0, parameters):
          diff = fabs(results[i] - certifiedValues[i]. ↩
              toDouble()[0])
          print  'db%d = %6g' % (i+1, diff)

        file.close()
        changeFolder(rootFolder())

newNote("ResultsLog").importASCII(dirPath + "/" + "results. ↩
    txt")
saveProjectAs(dirPath + "/" + "StRD_NIST.qti")
sys.stdout = saveout
fsock.close()
```

Chapter 8

Credits and License

QtiPlot

Program copyright:
2004-2009 Ion Vasilief ion_vasilief@yahoo.fr
2006-2007 Tilman Hoener zu Siederdissen thzs@gmx.net
2006-2007 Knut Franke Knut.Franke@gmx.de
Documentation copyright:
2004-2009 Ion Vasilief ion_vasilief@yahoo.fr
2006-2007 Roger Gadiou Roger.Gadiou@uha.fr
2006-2007 Knut Franke Knut.Franke@gmx.de

8.1 GNU Free Documentation License

Version 1.1, March 2000

8.1.1 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.

8.1.2 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.

8.1.3 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.

8.1.4 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.

8.1.5 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.

8.1.6 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."

8.1.7 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.

8.1.8 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.

8.1.9 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.

8.1.10 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.

8.1.11 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

Appendix A

Installation

A.1 How to obtain QtiPlot

The QtiPlot home page can be found at http://soft.proindependent.com. Updates and news can be found there. QtiPlot is distributed as a package with sources which have to be compiled. Precompiled packages for most Linux distributions, Mac OS X (>= 10.4) and for Windows can also be obtained from the same address.

A.2 Installation from binary packages

Ion Vasilief, the founder and main developer of QtiPlot, distributes precompiled packages for Windows, Mac OS X (>= 10.4) and for the Linux .deb based distributions: Debian, Ubuntu, etc... Generic Linux binaries, suitable for all Linux distributions, are also available. Several types of binaries maintainance contracts are available, please check the pricing details, or request an invoice. The fees include technical support and access to all updates available during one year.

On Windows, you need to download the qtiplot-0.9.7.8.exe file. To install, open a Windows Explorer window, double click on the 'qtiplot-0.9.7.8.exe' icon and follow the installation instructions.

The package provided for Mac OS X is universal and can be installed on both PowerPC and Intel Mac Pro systems. You can download it from here: qtiplot-0.9.7.8.pkg.zip. Before installation, you need to uncompress it, then double click on the 'qtiplot-0.9.7.8.pkg' icon and follow the installation instructions.

To install on Linux Debian: download the package file qtiplot_0.9.7.8_i386.deb, then login as "root" and type:

```
dpkg -i qtiplot_0.x.x_i386.deb
```

A.3 Compilation and Installation from sources

A.3.1 Requirements

If you want to build QtiPlot from sources, you need the following libraries:

Qt (>= 4.5.0) QtiPlot only works with the 4.5.x versions of the Qt library.

Qwt You must use the slightly modified version of the Qwt 5.2 library shipped with QtiPlot in the "3rdparty/qwt" folder.

QwtPlot3D A modified version of the QwtPlot3D library is used to create the 3D plots. It is included in QtiPlot ("3rdparty/qwtplot3d" folder), so you don't need to install it separately.

muParser The version of the muParser library used by QtiPlot is 1.32.

liborigin2 The last version of liborigin2 allows to read OriginLab 7.5 project files. It is included in QtiPlot ("3rdparty/liborigin" folder), so you don't need to download and install it.

BOOST C++ libraries liborigin depends on the BOOST C++ libraries (>= 1.33.0), so you need to download and install them separately.

GSL In order to build QtiPlot you also need the GSL library.

zlib For the compression of project files QtiPlot also uses the zlib library.

libpng For the export of 3D plots QtiPlot also uses the libpng library.

libxls For the import/export of Excel .xls files QtiPlot also uses the ExcelFormat library.

QuaZIP For the import of ODF spreadsheet .ods files QtiPlot uses the QuaZIP library.

Finally, if you want Python scripting support, you will also need to download and install: Python 2.x, SIP, and PyQt v4. All three of them are included in many Linux distributions.

A.3.2 Linux and Mac OS X

Download the qtiplot archive (.tar.bz2 or .zip file) and decompress it. Then open a terminal (console) window and go to the main folder of the decompressed archive (which should be named qtiplot-0.9.7.8). Create a build.conf file. You can use the provided example for this:

```
cp build.conf.example build.conf
```

You must make sure that QtiPlot is able to find the third party libraries it depends on. For this you have to edit the build.conf file. Once this is done, go back to the main directory qtiplot-0.9.7.8 and type:

```
qmake qtiplot.pro
make
```

to install QtiPlot, you must login as root and type:

```
make install
```

You can then logout from root and launch the application:

```
qtiplot
```

A.3.3 Windows

If you want to compile the application from sources you need to download and install the the MinGW compiler, the only one supported by the Qt Open Source edition on Windows. Then you must proceed exactly as on Linux/Mac OS X systems, following the above indications.

Chapter 9

Frequently asked questions

Q: *How can I visualise data from a text file?*
A: Go to the File menu and select the Import -> Import ASCII... command.

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.

Q: *How can I export a text file?*
A: Go to the File menu and select the Export ASCII 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 Data menu and use the Select Data Range command. Click in the plot window and use the 'Up' and 'Down' arrows keys to select the curve to analyse. 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 menu and select the The Fit Wizard... command. Define the function (myFunction=...), enter the initial guesses for the parameters, 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 analyse and select the option 'View pixel line profile' from the popup 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.

Q: *How can I make a graph that has one x-axis, but two y-axes with different orders of magnitude?*

A: Select at least two columns that you want to display and use the Double-Y command.

Index