The QtiPlot Handbook

COLLABORATORS			
	TITLE : The QtiPlot Handbook		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Ion Vasilief and Stephen Besch	22 February 2011	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

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 chapter describes the main concepts and terms which are used in QtiPlot.

-The second chapter 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. Plots can be produced from data sets stored in tables or from analytical functions.

The project was created by Ion Vasilief in 2000. Ion was the only programmer from 2000 until 2005. Since 2006, new contributors have joined Ion, and the project is hosted by BerliOS Developer. 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: Plots created from data sets, and the tables owning that data, are interconnected. When any table is modified, all objects in dependent plots (curves, axes scales, legends) are automatically updated. For example, deleting a table, or perhaps only some of the columns, will automatically remove all the corresponding curves from dependent plots. Plots can be exported in several graphic formats (eg: jpeg, png, bmp, pdf, etc) and inserted as images in documents or presentations.

All settings for a complete set of tables, matrices and plots can be saved in a project file having the extension ".qti". These project files may be opened using the command line, the File menu, or by using the *Open project* icon from the File toolbar.

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 file. They can be visualized at any time using the Results log command and can be deleted from the project file via the Clear Log Information command.

When the application is launched, a new untitled project file is created consisting of a grey main window (the workspace) which may initially contain an empty child window, depending on your preferences. The type of this initial child 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 at least one data container. Either empty tables or matrices may be created manually (New -> New Table command) and then filled with data, or they may created 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 by using either 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, minimizing, 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 table with the ASCII data from the file, and a 2D plot of all columns as a function of the first column in the file. The user is responsible for properly formatting the ASCII file. The file will be read and interpreted using the current settings from the Import -> Import ASCII... command dialog. The default values of these settings 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,
- spaces at the end of the lines are not removed,
- 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 mode!

1.3 General Concepts and Terms

Several plots and all the data related to these plots can be saved 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, plus a table and a plot window. Other windows are either docked or hidden.

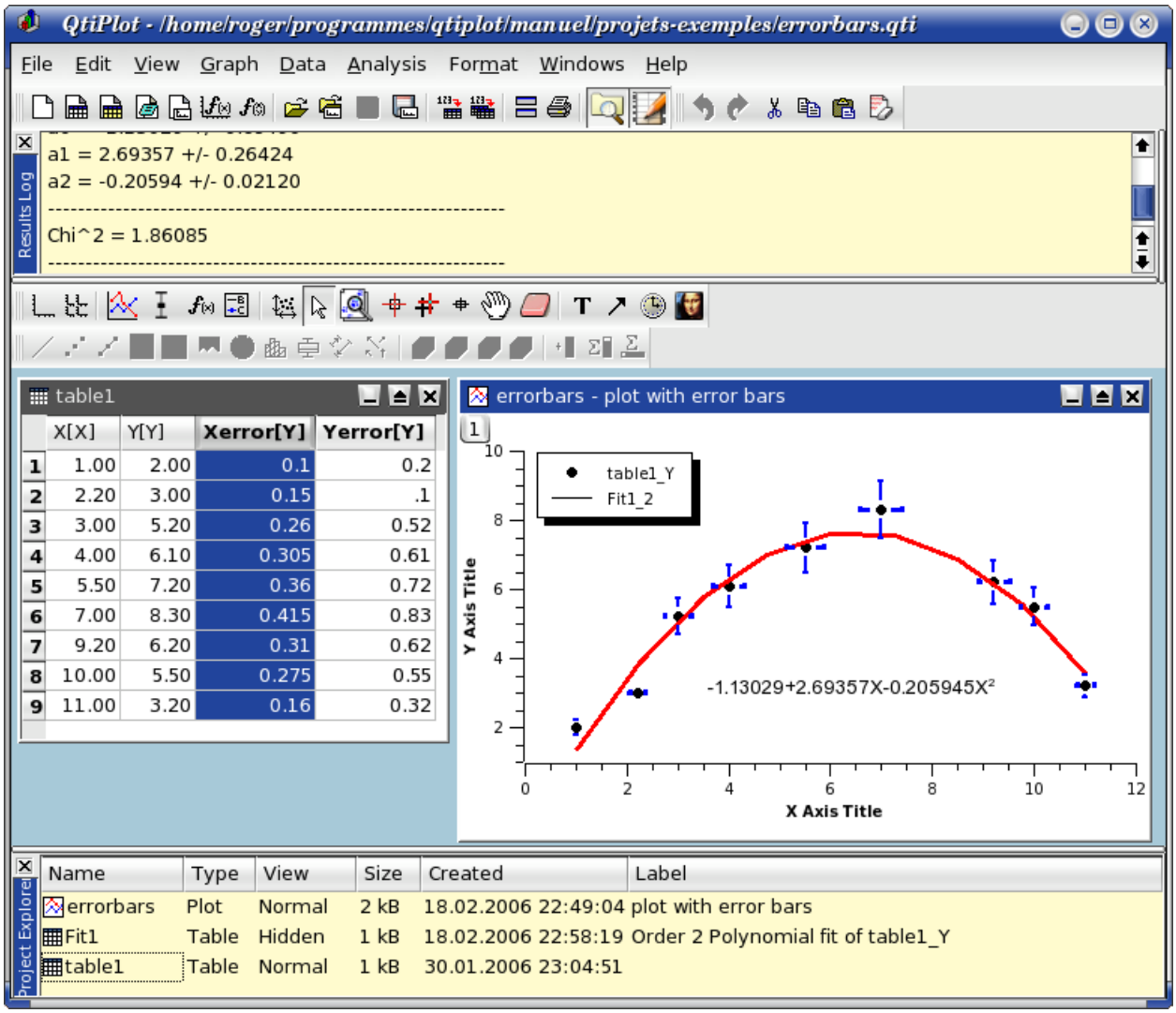


Figure 1.1: A typical QtiPlot session

General note on MDI style windows. QtiPlot uses a Multiple Document Interface (MDI) style for its sub-windows (for example graph and table windows, etc.). This is a convenient mechanism for placing sub-windows on a single parent window (the project window). Such collections of windows are then handled as a group when dragging or minimizing the main window. However, the behavior of maximized sub-windows is one feature of the MDI interface that may cause some confusion at first. As would be expected, sub-windows maximize to the size of the main window's workspace rather than to the size of the screen, but the default for maximized sub-windows is to have no title bar. As a consequence, there are no control boxes attached to the window, leaving the (incorrect) impression that once maximized, control boxes can no longer be used to minimize, normalize or close the sub-window. However, control boxes for a maximized sub-window are still present, they have just been moved to the extreme right hand side of the main window's menu bar. Since only one sub-window can be maximized at a time, there is no ambiguity regarding which sub-window this set of control boxes will operate upon. Finally, as a reminder of which sub-window is maximized, the Name and label of the maximized sub-window are appended to main window's title as:

"QtiPlot - ProjectName - [WindowName - WindowLabel]"

There are numerous commands available in QtiPlot. The specific subset of commands available depends 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 the set of commands relevant to a given element by activating the context menu with the right button of the mouse when the mouse pointer is floating over the chosen element.

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

A Table A table is a spreadsheet like object which can be used to store the data you are entering. The table is contained in its own window (the Table Window). It can be used to perform some calculations and statistical analysis of that data. In each table, columns can be labeled 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 using the New -> New Table command. There are then several ways to fill the table with data. If you want to read your data from an ASCII file, you can import it from the file into a table using the Import -> Import ASCII... command. You can also manually enter each value from the keyboard. Finally, you can fill the table with the results of evaluating a mathematical function using the (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 is created using the New -> New Matrix command. If you want to read matrix data from an ASCII file, you can import the data from the file into a table using 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 a matrix with the results of evaluating 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 *layers*. A layer consists of axes, text items, graphics, and a single *plotting area* bounded by the axes lines. One or more *curves*, generated from data or functions, are placed into the plotting area to create a *plot*. Layers and their contained plots can be arranged in many ways to build matrix of plots. Throughout this document, the term *plot window* is used as a synonym for a graph.

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

Curves can be added to a plot in several ways. You can select data from tables or matrices to generate the curve, or, create a curve from a function 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 powerful 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 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 previous calculations.

The Project Explorer This window is used to list all the windows contained in a project. The Project Explorer gives quick access to all elements of a project, hidden or visible. It can be used to perform some operations on the listed windows such as hiding a window, renaming a window, etc.

Since version 0.8.5, a project file can include several independent projects. In this case, the containers of each project are stored in different folders.

1.3.1 Tables

When working with data, tables are the main focus of QtiPlot. Fundamentally, a table is simplified spreadsheet contained in a Window which can be used to control, edit, and convert data. Tables are also highly customizable: all colors and font preferences can be set using the Preferences... command of the View menu, and you can resize a table in terms of rows and columns using the Table menu with Rows or Columns.

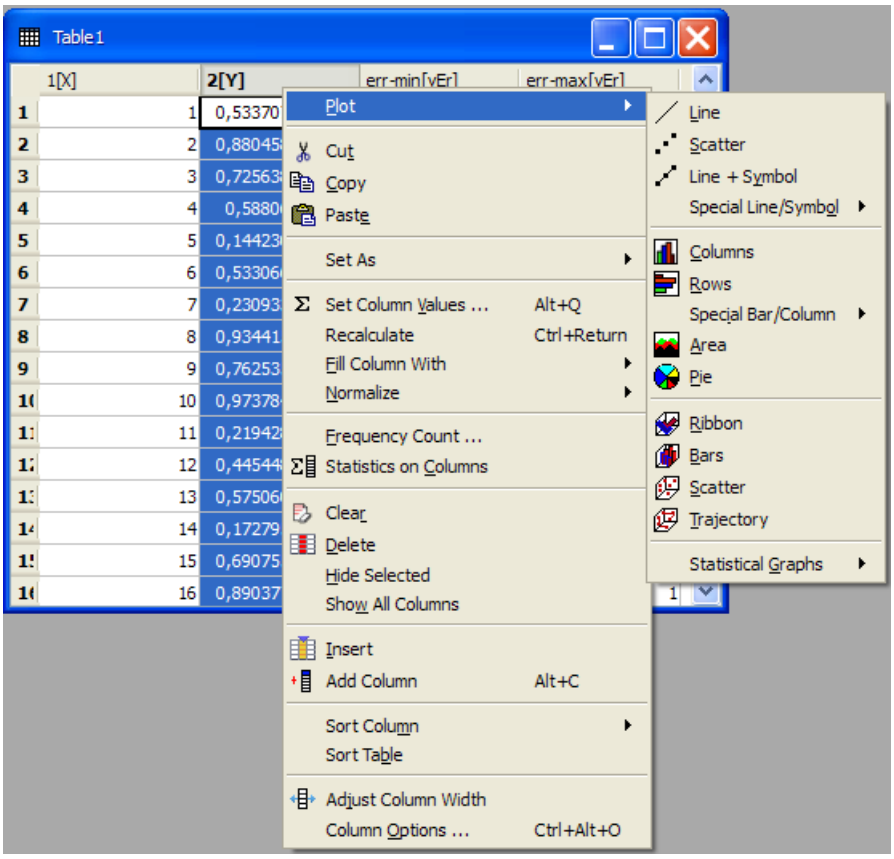


Figure 1.2: The QtiPlot table

Every column of a table has a label, and can be assigned a format: numeric, text, date or time. Each column can also have one of the following flags set: X, Y, Z, X-error, Y-error, label, or none (i.e., a simple column without any special flag). X flagged columns are the abscissae while Y flagged columns are the ordinates used when creating a 2D plot from data. A column must have either the X or Y flag set to be available for use in a 2D plot. The X-error and Y-error columns can be used to add error bars to a curve in a 2D plot. Flags can be changed using the Column options dialog. To reach this dialog, simply double-click on the column label or use the Column Options... command from the Table menu.

A table column is selected by left clicking on it's label. Multiple columns are selected in one of 2 ways. First, if the columns are adjacent, it is most convenient to left click on the first desired column's label and, while holding the left mouse button down, drag the mouse pointer over the labels of the column you wish to select. Second, in the case where desired columns are not adjacent, you can select additional columns by keeping the Ctrl key pressed while left clicking on the desired column's label. This also allows you to deselect specific columns. You can select all the columns of a selected table by pressing (Ctrl+A).

You can perform various operations on selected columns : fill with data, normalize, sort, view statistics and finally, generate curves from your data. All these functions can be reached by right clicking on the column label or by using the Table menu.

All other table functions: rename, duplicate, export, print, and 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 tables or between a table 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 also export the data from a table to a text file using the Export ASCII command.

1.3.2 Matrix

The matrix is a special table which is used for data which depends on two variables. This special table can be used to create 3D plots as well as 2D image/contour plots via the Plot 3D menu and the 3D plot toolbar. One difference between a table

and a matrix is that matrices may function in one of two modes: they can display data in 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 functions like: 90 degrees rotation, horizontal and vertical mirroring, etc.

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

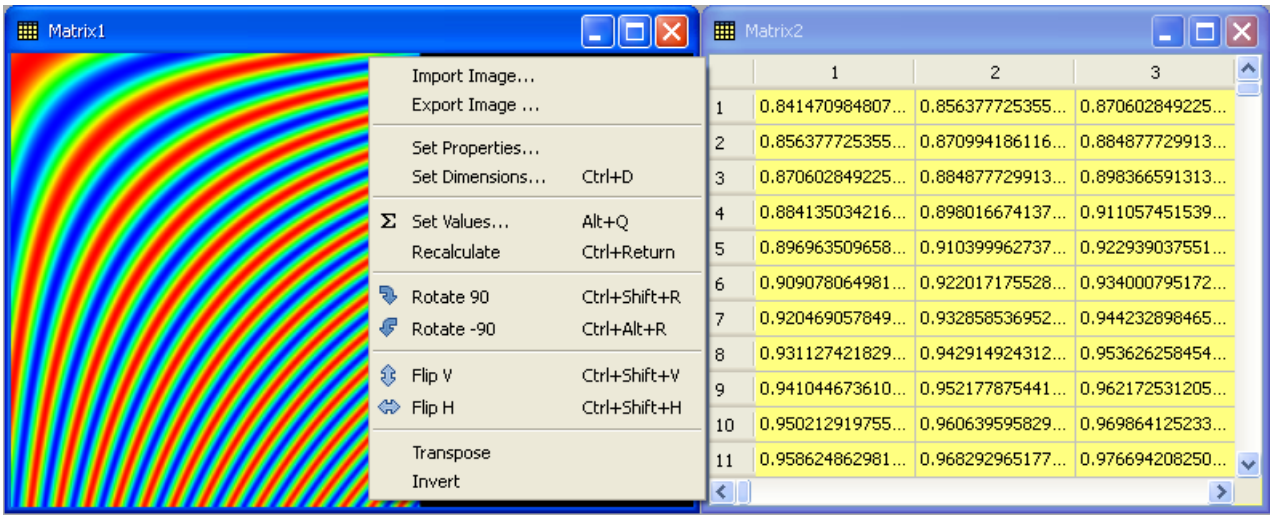


Figure 1.3: The QtiPlot matrix

The values which are stored in a matrix can be generated from a function of the form z=f(i, j, x, y) with the Set Values... command, i and j being 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 (that is, a graph), provides a container for plotting data. It contains one or more layers, which are the main containers of a graph. Each layer contains a plotting area into which curves are placed when creating a plot. Each layer 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.

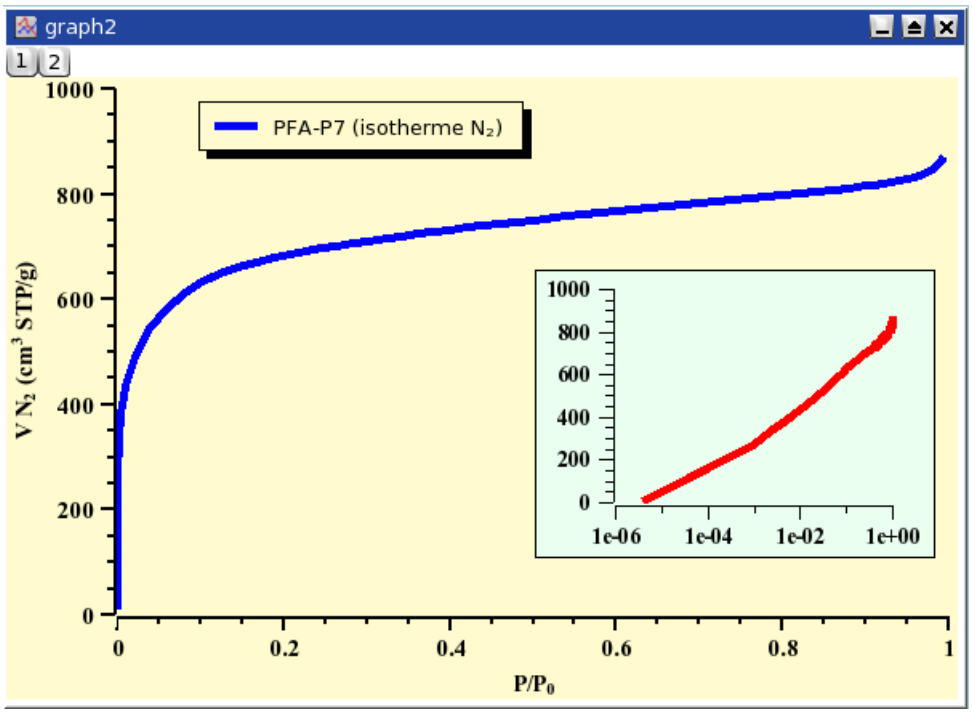


Figure 1.4: An example of QtiPlot 2D graph

Each layer can be activated by clicking on its corresponding gray button 1 2 in the top-left corner of the window.

Some graph elements can be accessed by a double click on an element in a layer. These 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: this will open the Text Options Dialog which allows customizing 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 you to customize the geometry and the position of the image.

A left click on a layer element selects it. You can deselect any element by pressing the *Escape* key. A right click on a layer element pops-up a context menu allowing quick access to its properties dialog. Last but not least, you should know that QtiPlot provides multiple selection for objects in a 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 powerful 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 convenient keyboard shortcuts. For information on expression syntax, supported mathematical functions and how to write scripts, see here.

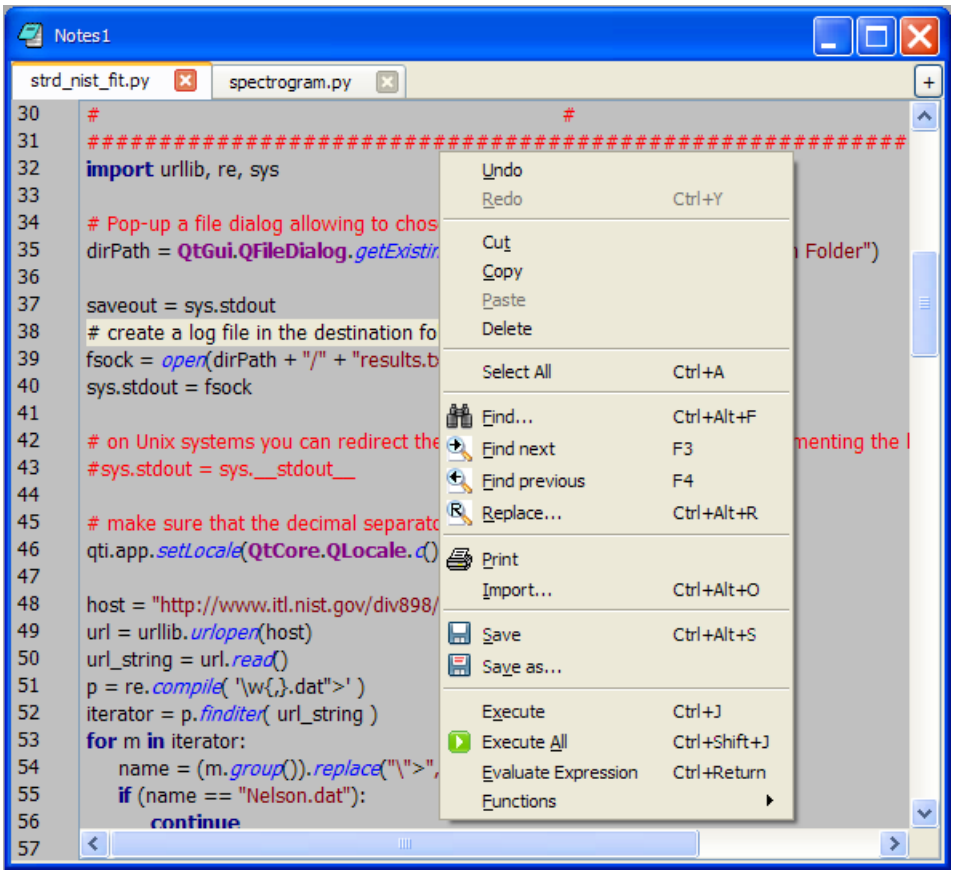


Figure 1.5: The QtiPlot Note Window

Note windows provide powerful text editor functionalities, particularly helpful when writing scripts: customizable Python syntax highlighting, line number display, find and replace text, and autocompletion suggestions for words having more than two characters. You can manually trigger autocompletion by 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 has been done in the project. It panel contains the results of all the correlations, fittings, etc.

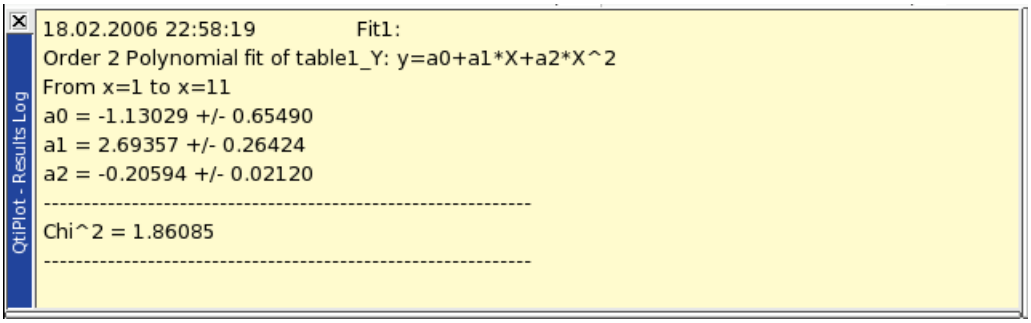


Figure 1.6: The QtiPlot Log window

1.3.6 The Project Explorer

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

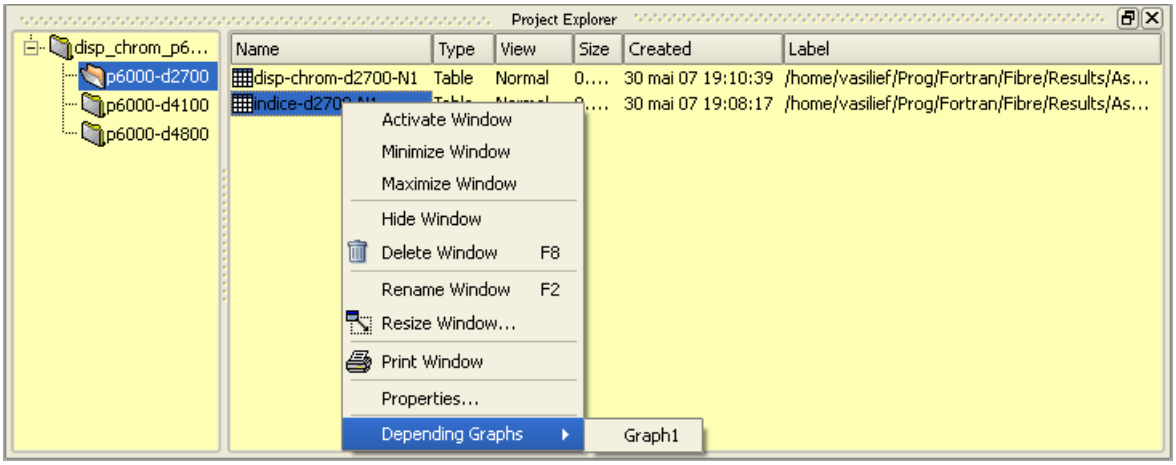


Figure 1.7: The QtiPlot Project Explorer

It gives an overview of the structure of a project and allows the user to perform various operations on the windows (tables, graphs, and notes) in the workspace: hiding, minimizing, closing, renaming, printing, etc. These functions can be reached via the context menu, obtained 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.

From the project explorer window, different objects can be organized into folders. When selecting a folder, the default policy is that only the objects contained in it will be shown in the workspace window. You can also display all the objects in 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 using the "Plot Designation" option found in the Column Options... command. Then you select the columns and use one of the commands in the Plot menu to plot the data.
- If you want to plot a function, you don't need a table at all. You can plot the function directly with the New -> New Function Plot command. This opens the corresponding dialog box where you define the mathematical expression of your function.
- These two methods can be combined by first defining a table, and then filling it with the results of evaluating your function. This is done with the Set Column Values... command. Then you select the columns and use one of the commands from the Plot menu to plot the data.

In each of these cases, QtiPlot will create a new graph with the plotted curve placed on a new layer. Data plots and function plots can also be added to an existing layer using either the New -> New Function Plot command command or by right clicking within the area of the desired plot to pop up the plot's Graph Menu, and then selecting Add...Add Function.

Once the plot is created, you can customize all the graphic items in the plot using commands from the Format Menu. You can add new items (text labels, lines or arrows, new legend, images) to 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 two methods for inserting your (X,Y) values into the table: you can type them directly from the keyboard, or you can read them from a file. Here we will use the first solution, refer to the Import -> Import ASCII... command to use the second.

The first step in this example is to create an empty project with the New -> New Project command from the File menu. You can also use the Ctrl-N key or the icon from the File toolbar. Next create a new table using the New -> New Table command from the File menu, the Ctrl-T key, or the icon from the File toolbar.

A newly created table has two columns (one for X and one for Y) and 30 rows. You can add rows and columns by selecting a row or a column and using the right mouse button. You can also modify the number of rows and columns with the Rows and Columns from the Table menu. Try setting the number of rows to 7, which will match the table shown below. Then enter the values as shown (you can of course use your own data). You should now have this table:

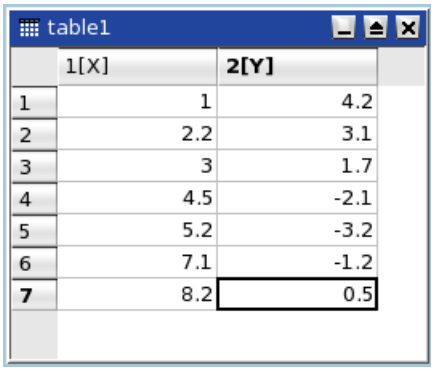


Figure 2.1: A simple 2D plot: the table.

You must next select the data to be plotted. To select the 2 columns of data just entered, left click on the title of first column and drag the mouse pointer over to the title of the second column while holding the left mouse button down. Now, with the columns selected, you can build the 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 in the plotting area of a new layer on a new graph. Default options are used for for all newly created elements. You can customize the default options with the preferences dialog. The default options will produce the following:

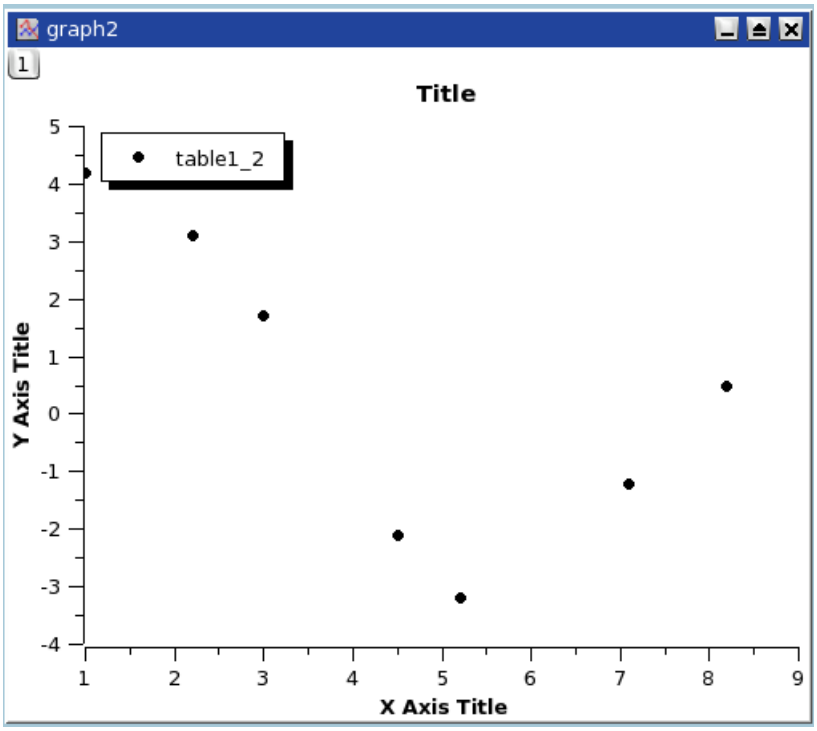


Figure 2.2: A simple 2D plot: the default plot.

You can now customize your plot and the elements of the parent layer. Double clicking on any point will open the Custom curves dialog, which is used to modify the plotted symbols. A double-click on any axis opens the general plot options dialog, where you can change scales, fonts for the axis labels, etc. You can also add grid lines on X or Y axes, etc. Finally, a double click on any text item (X title, Y title, plot title) allows you to change the text and its presentation. As an illustration, several changes have been made to the above plot. The final result is:

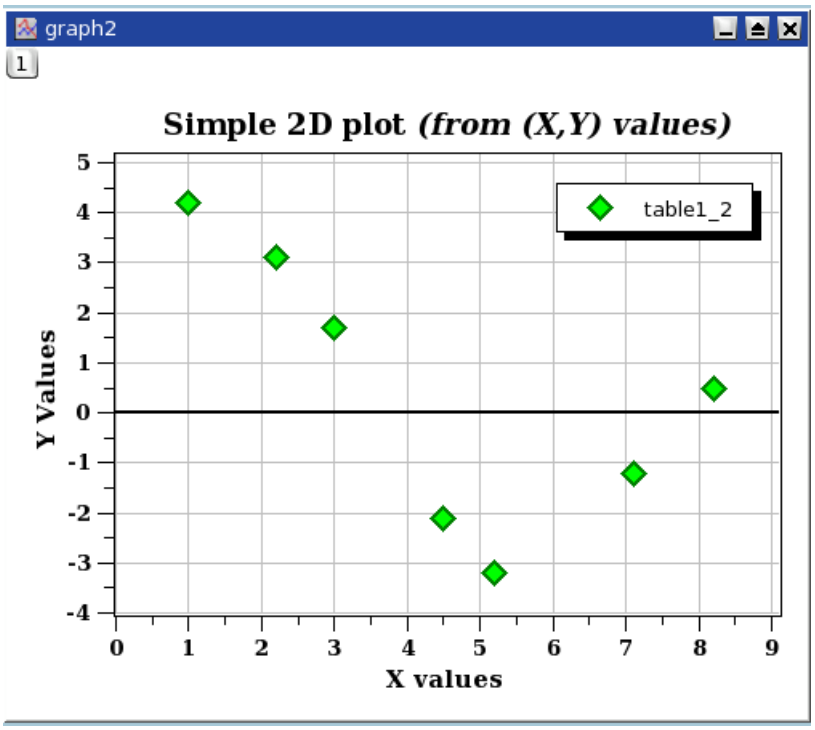


Figure 2.3: A simple 2D plot: the plot finished.

Finally, you should save your project in a '.qti' file using the Save Project command from the File menu or by typing the Ctrl-S key, or by clicking the icon from the File toolbar. Depending on your needs, you can export the plot in any of several standard image file formats using the Export Graph -> Current command from the File menu, or by entering the Alt-G key.

There are several types of curves which can be plotted from a table. They are presented in the Plot menu

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

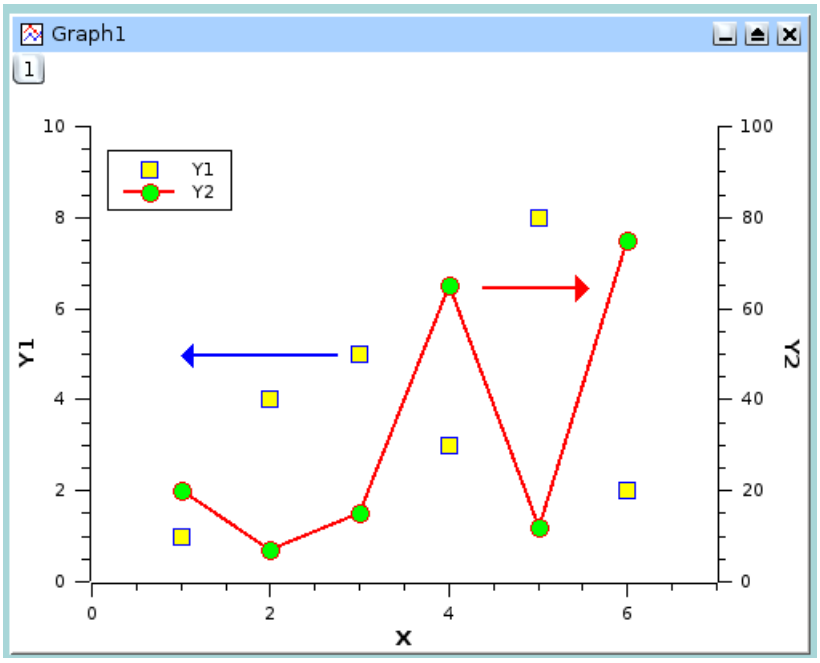
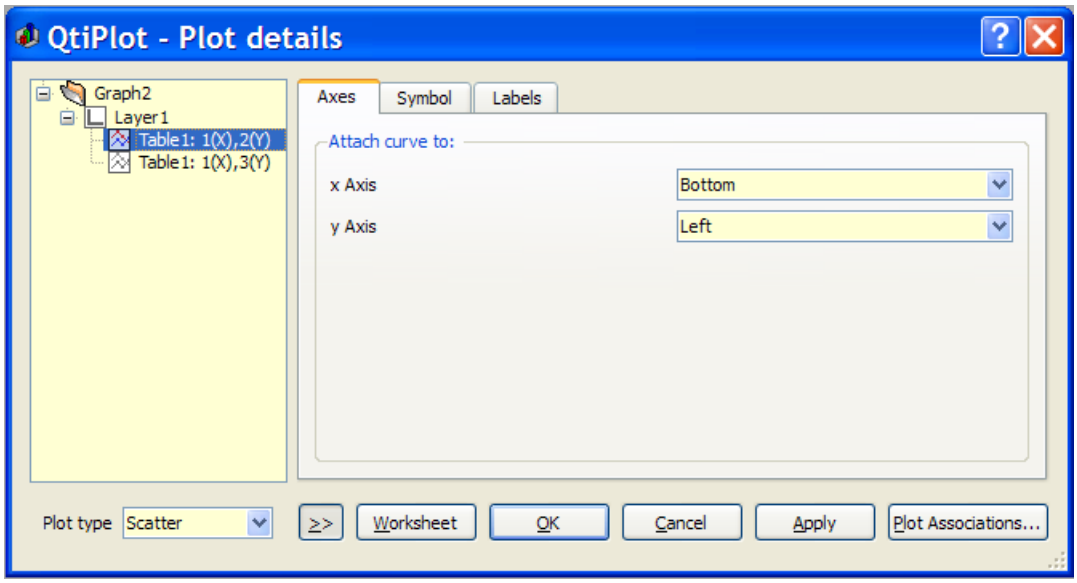


Figure 2.4: A 2D plot with two Y axes.

In addition to the customizations which have been already been described, for the figure above the axes used for each curve were defined using the Custom Curves Dialog, and two arrows were added with the Draw Arrow. Note that the table must be modified by the addition of a second column of Y data before the second curve can be drawn in the plotting area (using Graph Menu, and then selecting Add...Add/Remove Curve).



2.1.2 2D plot from function.

There are two ways to obtain such a plot: you can plot a function directly, or fill a table with the values calculated from a function and create the plot in the usual 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, click the icon in the File toolbar, or simply enter Ctrl-F.

This command will open the Add Function Curve dialog. You can then enter the mathematical expression of your function, the X range to be used for the plot, and the number of points in the X range. Besides classical Y=f(X) functions, you can also define parametric and polar functions.

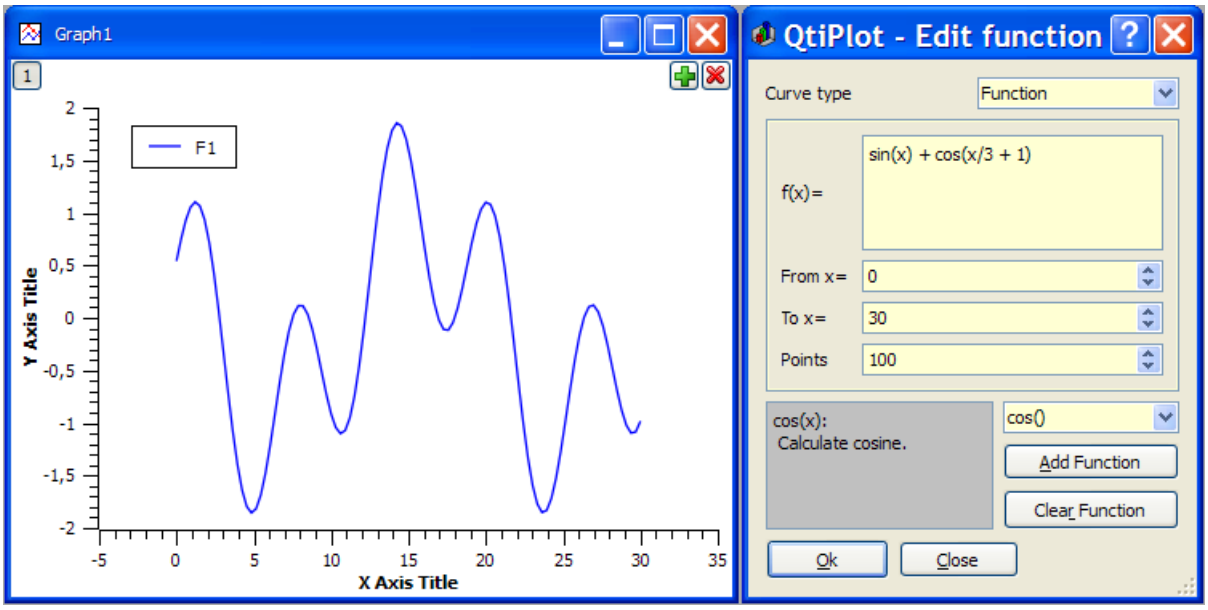


Figure 2.5: Direct plot of a function.

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

If you want to work not only with the plot but also with the resulting data, create a new table as explained in the previous section. Then fill this table with the values of the function evaluation using the Set Column Values... command.

Let's obtain the same plot as in the previous example. Create a new table (key Ctrl-T), select the first column and use the Set Column Values... command either from the context menu, or the Table menu. The row number can be used in functions by referencing the row number symbol, *i*. For a range of 0.01-30 in 300 steps (0.01 per step) enter the function expression *i/10* and use 300 rows. (Note that since row numbering starts at 1, to actually get the X range used in the last example (0-30 over 300 points), we would need to define the function expression as *(i-1)*30/299*.)

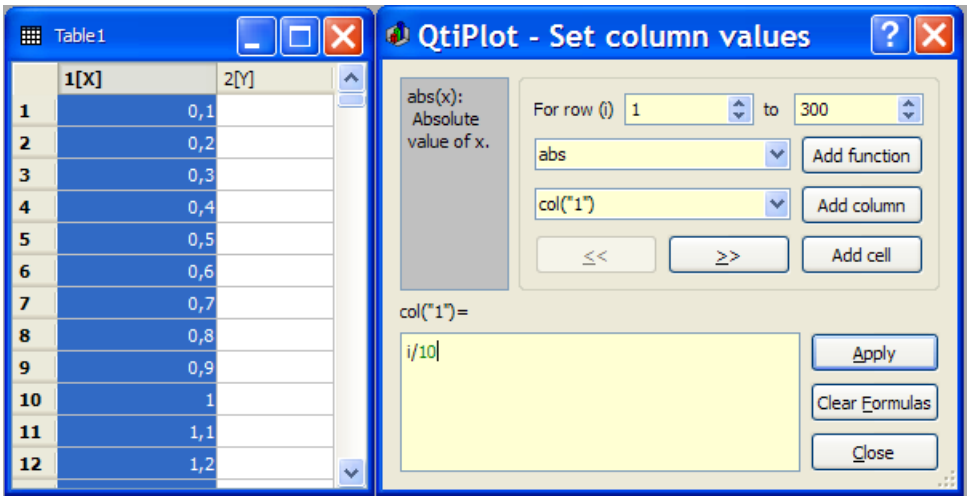


Figure 2.6: Function plot: filling of the X column.

The second step is to select the second (Y) column and use the Set Column Values... command to set up the function. The expression is a function of the X values (that is the first column) which is named *col(1)*. Enter *sin(col("1"))+cos(col("1")/3+1)* as the function and click apply to generate the values in the Y column.

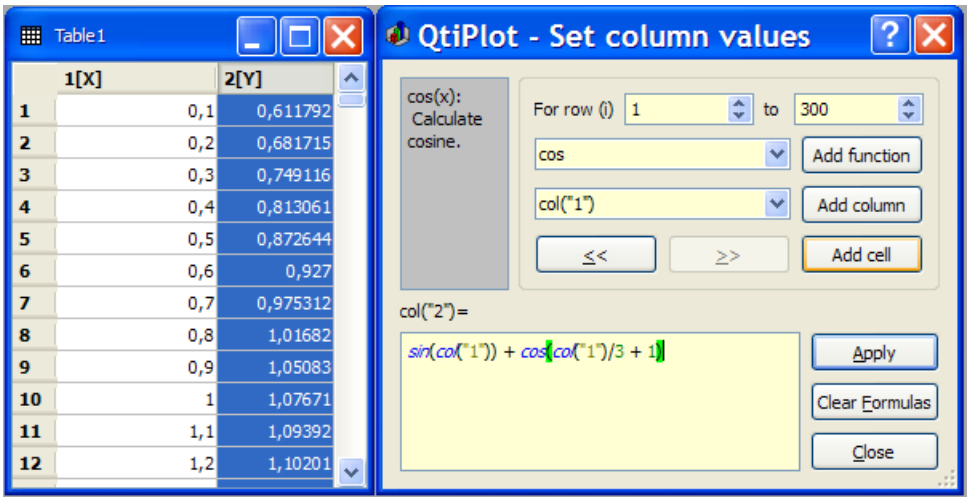


Figure 2.7: Function plot: filling of the Y column.

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

2.2 3D plots

3D plots are generated from data defined as Z=f(X,Y). As with 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 are defined as functions of the column and row numbers.

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

  – one by one from the keyboard,
  – by reading an ASCII file into 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 plot a function directly using the New -> New Surface 3D Plot command. This will open the corresponding dialog box where you 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 available plots.

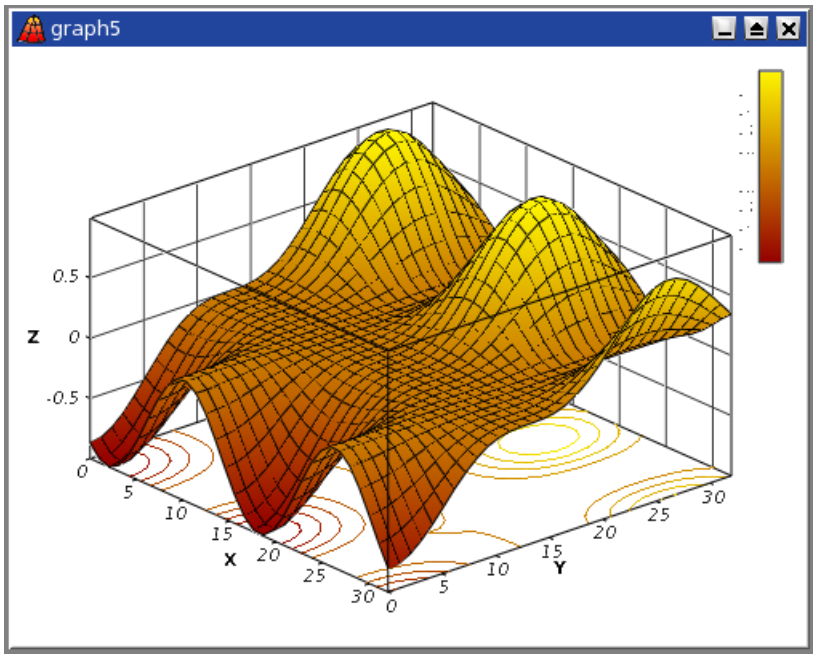


Figure 2.8: Example of a 3D Plots.

3D plots use OpenGL so you can easily rotate, scale and shift them with the mouse. Using the 3D plot settings dialog or 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. Use the New -> New Surface 3D Plot command from the File menu or simply enter Ctrl-Alt-Z. This will open the following dialog box:

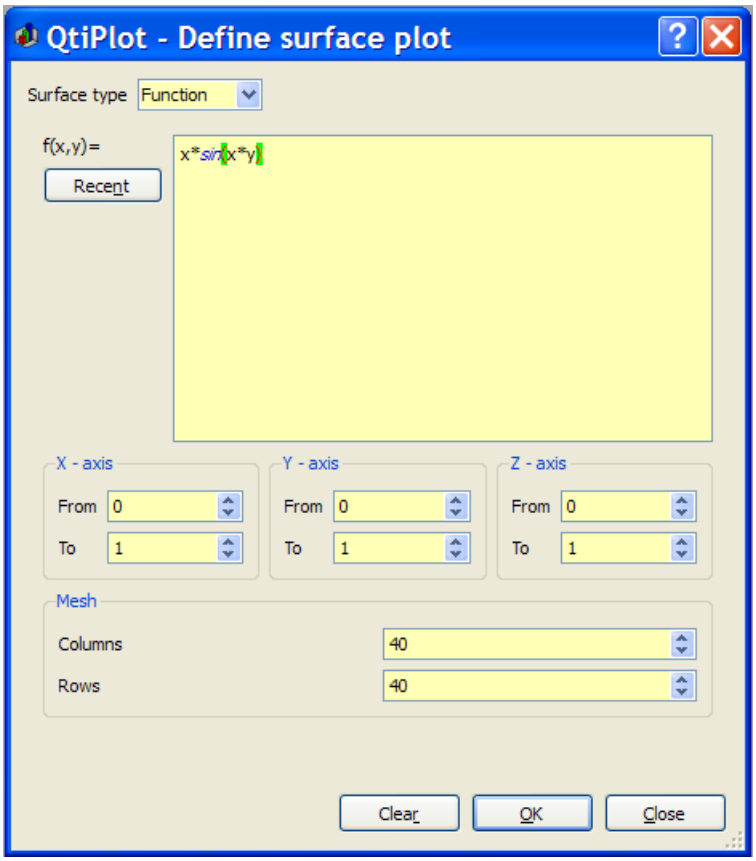


Figure 2.9: Definition of a new surface 3D plot

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:

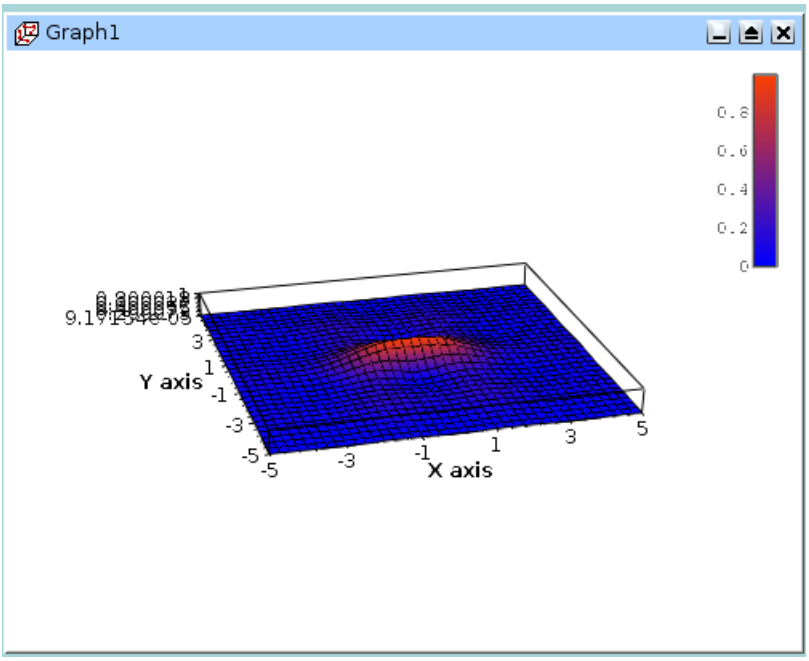


Figure 2.10: The 3D surface plot created using defaults

You can then customize the 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 commands of the 3D plot toolbar to add grids on the walls or to modify the style of the plot. The following plot illustrates some of the possible modifications:

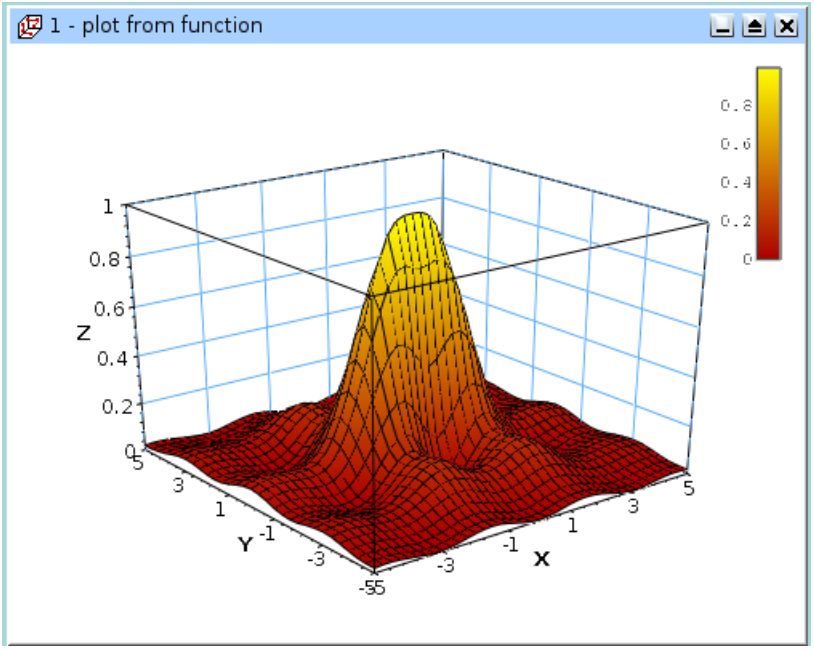


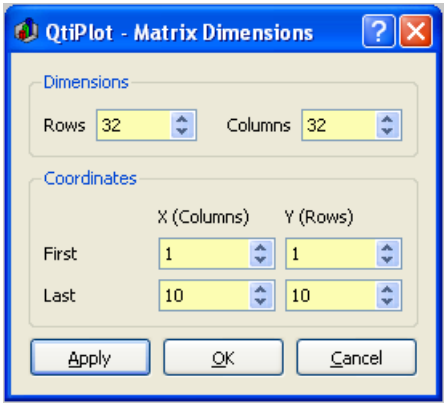
Figure 2.11: The 3D surface plot after customization.

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 evaluation of 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, so the function must be defined with row and column numbers (i and j) as 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 into a matrix with the Convert to Matrix command from the Table menu.

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

2.3 Multilayer Plots

Graph windows can contain multiple layers, each with different characteristics. Each layer has a corresponding, numbered button. A button appears pressed when its layer is the currently active layer. Only one layer is active at a time, and the plot tools (zoom, cursors, drawing tools, delete and move points) only operate on this layer. A layer is made active by clicking on it or on its corresponding button.

To arrange layers use the Arrange Layers dialog. You can add or remove layers with the Add Layer and Remove Layer 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 layer area).

You can resize and move a layer using the Layer geometry dialog (Select the Geometry tab in Plot... from the Format Menu). You can also arrange and resize layers by hand using the mouse. First, select the layer to be modified or moved and then select the layer's plot area by left clicking on a border line (an axis line) of the plot area. QtiPlot will draw a box outlining the plot area with drag handles at the corners and midpoints of the sides. The cursor will assume a shape based upon where it is located on the layer: a hand shape when inside the plot area, or a double ended arrow when over one of the drag handles. The cursor's function, which is invoked by pressing the left mouse button, is indicated by this shape.

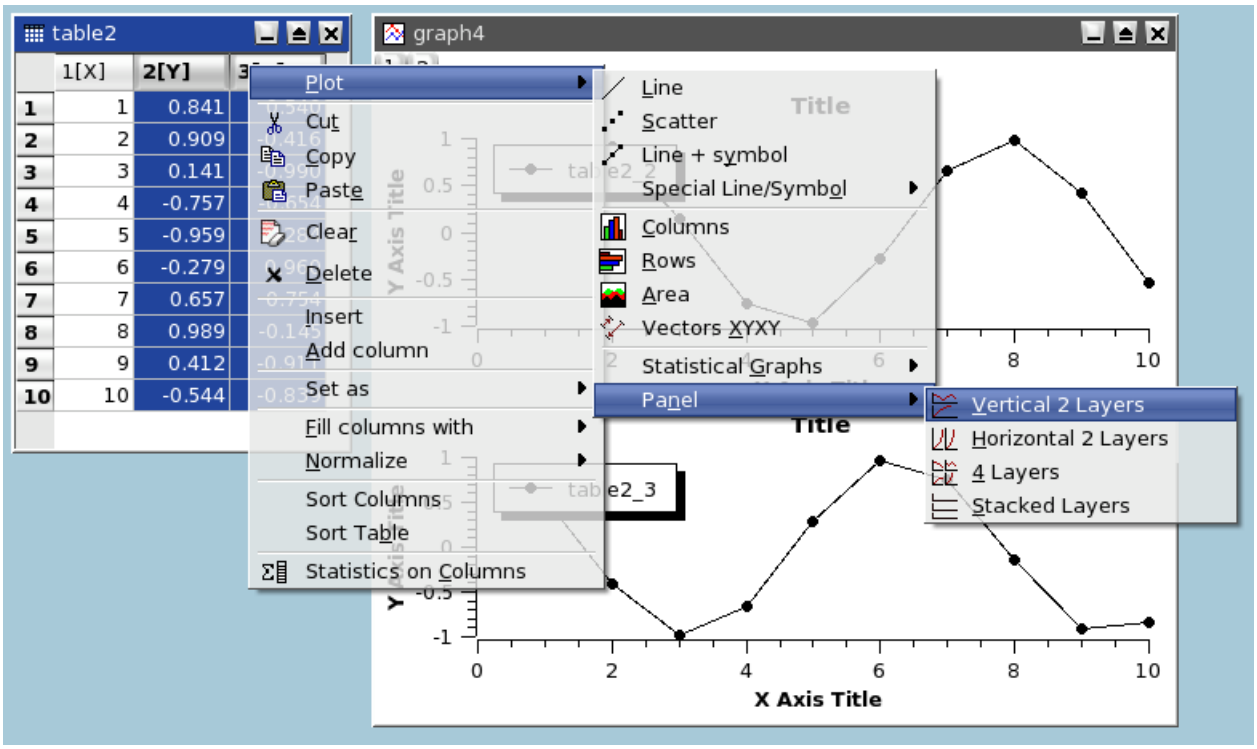
Pressing and holding the left mouse button when the cursor assumes the hand shape allows dragging the entire layer with the mouse. Releasing the mouse button will drop the layer at the new position. Pressing and holding the left mouse button when the cursor assumes one of the double ended arrow shapes allows dragging the corresponding border of the plot area, scaling the layer as needed. Releasing the mouse button will drop the border and apply the new scale value. Grabbing a midpoint handle moves only the corresponding border, while grabbing a corner handle allows simultaneous dragging of both corresponding borders. Dragging the corner handles does not preserve the aspect ratio of the layer.

You can also conveniently resize a layer using the mouse wheel in combination with either the Ctrl, Alt, or Shift keys. In order to use the mouse wheel, the desired layer must be selected, either by clicking in the layer or by using one of the layer selection buttons. It is *not* necessary to have the plot area selected, although the wheel functions will work in either case. The wheel functions work as follows: Pressing and holding the Ctrl key while rotating the wheel resizes the height, pressing and holding the Alt key while rotating the wheel resizes the width, pressing and holding the Shift key while rotating the wheel resizes both the height and width. Note that in this last case (Shift+Wheel), the aspect ratio of the layer is preserved.

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.

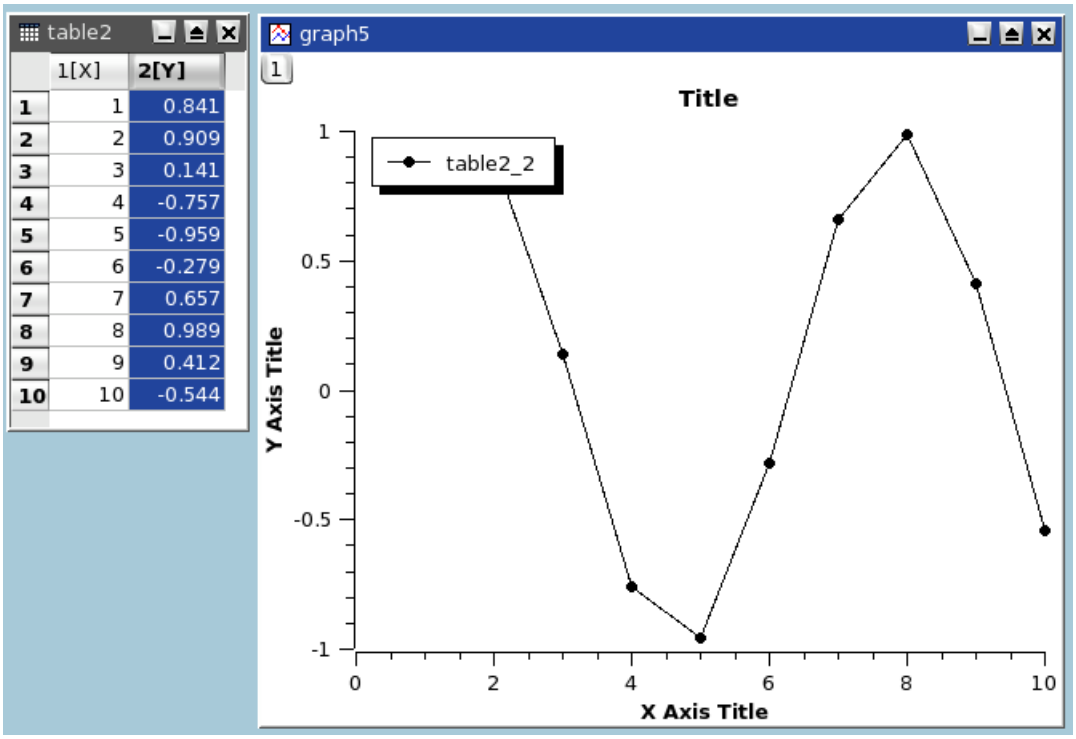


You can then customize the plots. If you want to change the arrangement of the panel, use the Arrange Layers from the Graph menu. In this case, keep in mind that each plot is in its own layer whose surface area is one half or one quarter of the window's surface area. So, if you want to share an element between two plots (for example a text label), you need to add it in a new layer (see the Add Text for more details).

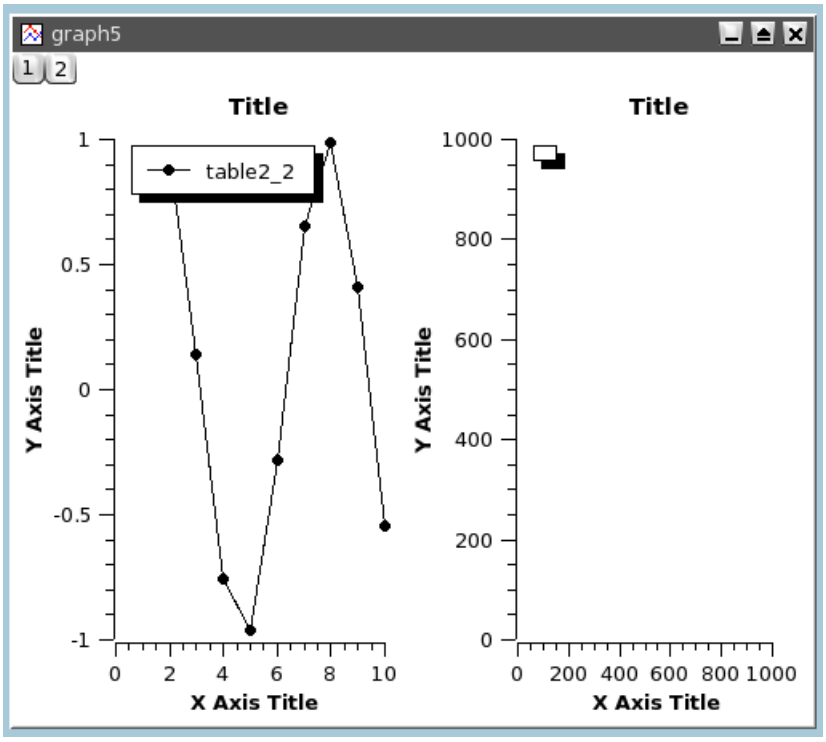
2.3.2 Building a multilayer plot step by step

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

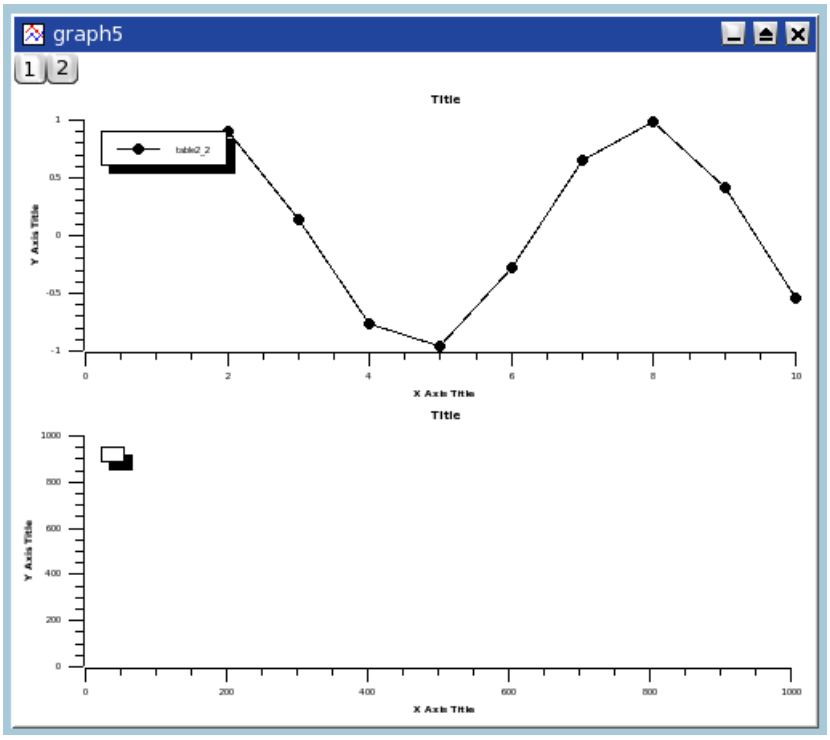
The first step is to build your first plot (for example from two columns of a table). Start by creating a standard graph window:



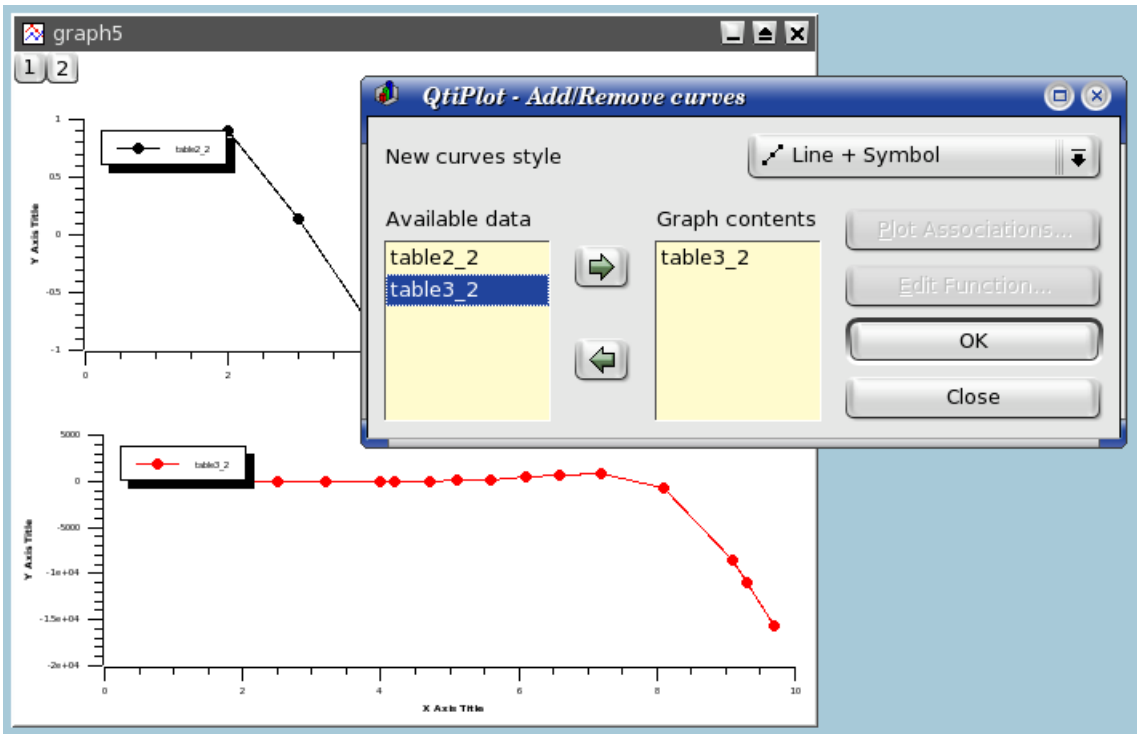
Then, select the plot window and use the Add Layer from the Graph menu. This will activate the Add Layer dialog. If you choose "Guess" you will obtain a panel with two columns, if you choose "corner" you will obtain two superposed layers, you can then modify these two layers.



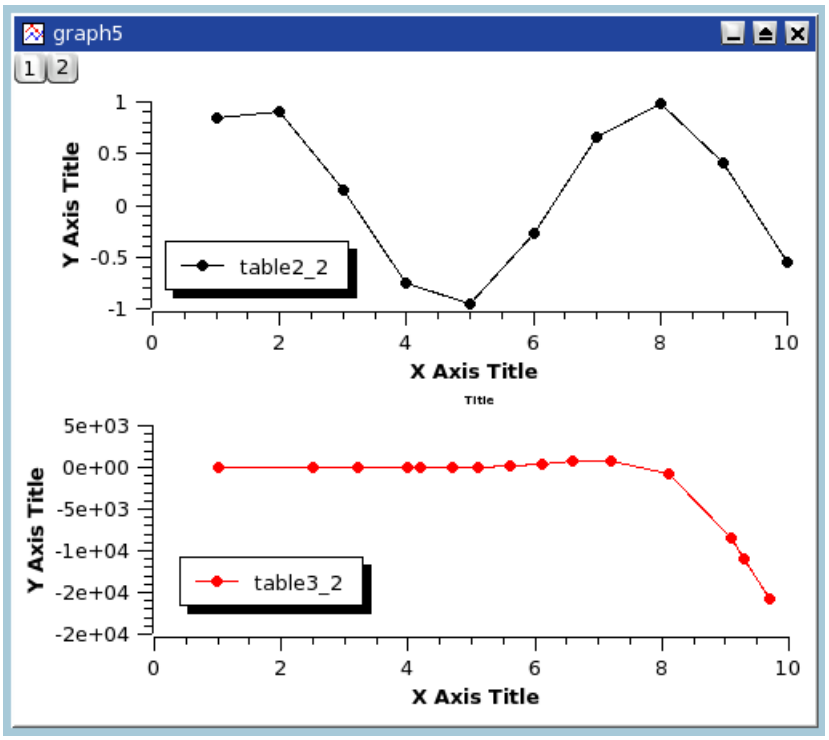
If you want to build a panel with two rows, you can use the Arrange Layers to automatically rearrange the layers.



Then select the second (empty) 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 again rearrange the two plots by using the Arrange Layers a second time.



Chapter 3

Command Reference

The active items appearing in a menu depends upon which project window is active. For example, if the active window is a table, then all the items related to table functions are enabled and the others are automatically disabled.

3.1 The File Menu

Many of the commands from the File Menu are also linked to corresponding icons in the File Toolbar. Clicking one of these icons will directly execute the linked command.

3.1.1 File-> New ->

3.1.1.1 New -> New Project (Ctrl-N)

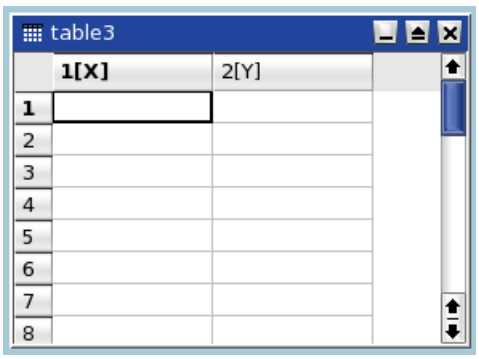
Creates a new QtiPlot project file. If another project is already open and has been saved at least once, it will be closed before the new project is created. If another project is open but has never been saved, a dialog will be opened to ask if the current project should be saved.

3.1.1.2 New -> New Folder (F7)

Adds a new folder to the project. The new folder is added to the current folder.

3.1.1.3 New -> New Table (Ctrl-T)

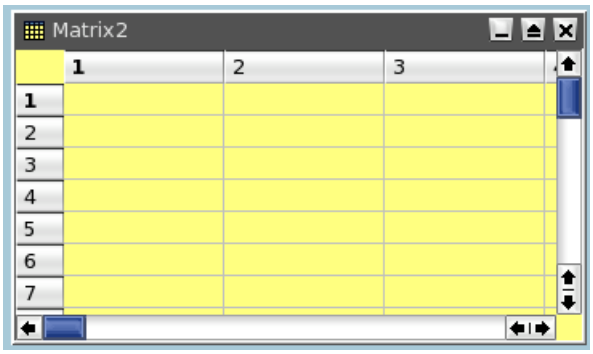
Creates a new (empty) table and adds it to the project. The empty table will have 30 rows and 2 columns. The number of rows and columns can be changed with the Rows and Columns of the Table menu.



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

3.1.1.4 New -> New Matrix (Ctrl-M)

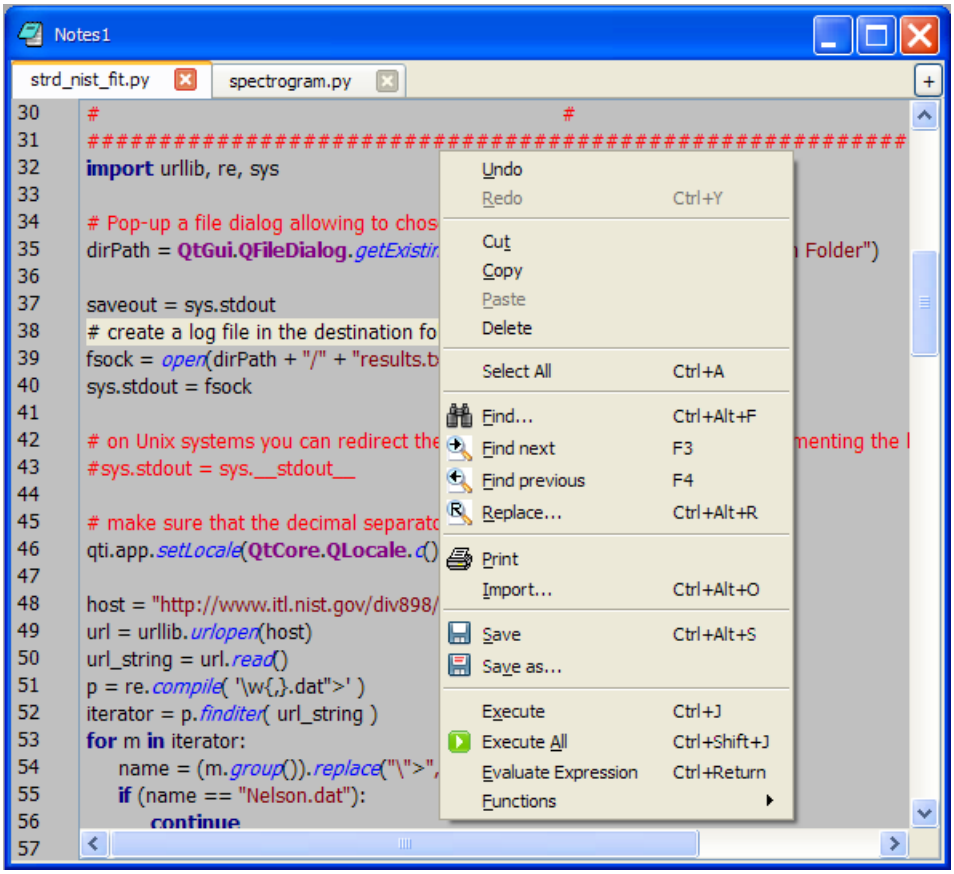
Creates a new (empty) Matrix and adds it to the project. The empty matrix will have 32x32 cells. These dimensions can be changed using the Set Dimensions... command of the Matrix menu



See the matrix section for more details.

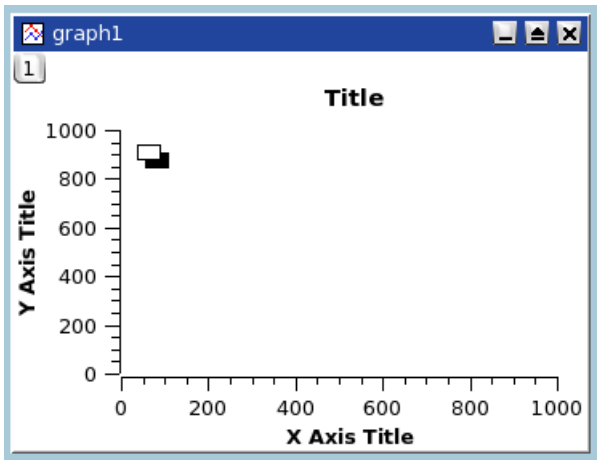
3.1.1.5 New -> New Note

Creates a new note window and adds it to the project. A note is a simple text window which can be used to add comments to the current project.



3.1.1.6 New -> New Graph (Ctrl-G)

Creates a new 2D plot, that is, a graph window with a single, empty layer, and adds it to the project. Current defaults are used to create the layer, which is just a framework into which you add curves with the Add/Remove Curves... command.



3.1.1.7 New -> New Function Plot (Ctrl-F)

Opens a dialog which is used to create a 2D plot by specifying an analytical function. See the 2D plot section of the tutorial for a general overview of this function.

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

3.1.1.8 New -> New Surface 3D Plot (Ctrl-Alt-Z)

Opens a dialog which is used to create a 3D plot by specifying an analytical function. Only Cartesian coordinates are available. See the 3D plot section of the tutorial for more detail on this function.

3.1.2 File -> Open (Ctrl-O)

Opens an existing QtiPlot project file (.qti). If your project has been saved 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 *Origin* software.

3.1.3 File -> Open Excel

Opens a file dialog permitting you to select an Excel file. When a file is selected and opened, QtiPlot creates a new table for each sheet in the file and reads the spreadsheets into the tables. If the file contains graphs, QtiPlot will also import them, but only if you have Excel installed on your machine.

3.1.4 File -> Open ODF Spreadsheet

Opens a file dialog permitting you to select an OpenOffice spreadsheet file. When a file is selected and opened, QtiPlot creates a new table for each sheet in the file and reads the spreadsheets into the tables.

3.1.5 File-> Open Image File (Ctrl-I)

This command adds a new graph window to the QtiPlot project and loads an image file into it. The image can be resized and moved around in the graph window if desired. It can also be copied and inserted into another 2D plot with a result similar to that obtained using the Add Image. An image can also be used to generate an intensity matrix (see the Import Image... command).

3.1.6 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 append projects which have been built with *Origin* software.

3.1.7 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 no longer exists or has been moved, an error message will pop-up and the filename will be deleted from the list.

3.1.8 File -> Close

Closes current project, without quitting the application.

3.1.9 File-> Save Project (Ctrl-S)

Saves the current project. If the project hasn't been saved yet (an "untitled" project), a dialog will open, allowing you 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 includes large tables, it may be useful to save the project in a compressed file format. The free zlib library is used to save files in gzip format (.qti.gz).

3.1.10 File-> Save Project as... (Ctrl-Shift-S)

Saves the current project under a file name different than the current name.

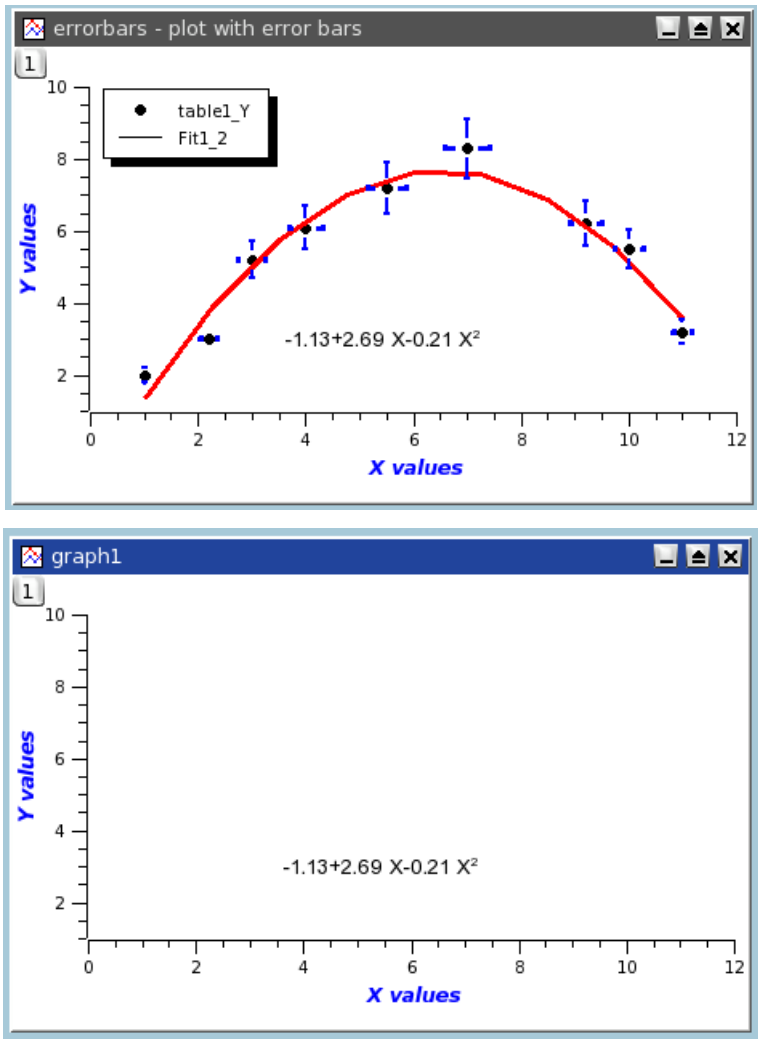
3.1.11 File-> Save Window as...

The **Save Window as...** command allows you to save tables and graphs from from one Qti project into a newly created project file. The command opens a standard file-save window in which you select the new project's name and location. Projects may be saved in either compressed or uncompressed form. If a graph window is selected, the new project will contain the graph and it's associated tables/matrices. If a table, matrix or note is chosen, the new project will contain only the selected table, matrix or note. In this case, dependent graphs are not included in the new project.

3.1.12 File -> Open Template

Opens an existing QtiPlot template file (.qpt). This command will create a new graph window with one or more empty layers created using the same graphical parameters (window geometry, fonts, colors, etc). as the layers in the saved template.

The first figure below is a graph which was saved as a template. The second figure is the graph with a new, empty layer created using the **Open Template** command to load the saved template file.



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

3.1.13 File -> Save as Template

Save the active graph as a QtiPlot template file (.qpt). The resulting template will contain all of the layers from the graph, including images, text labels (axes, etc), and any graphical parameters, including the positions and sizes of the layers. Plotted data, the style used to draw curves, and any associated scales are not saved in the template.

3.1.14 File-> Print (Ctrl-P)

Prints the active plot. A print dialog is opened where you can select the printer, different paper sizes, etc.

3.1.15 File-> Print Preview

Displays a print preview for the active window. You can use this dialog to print the previewed window.

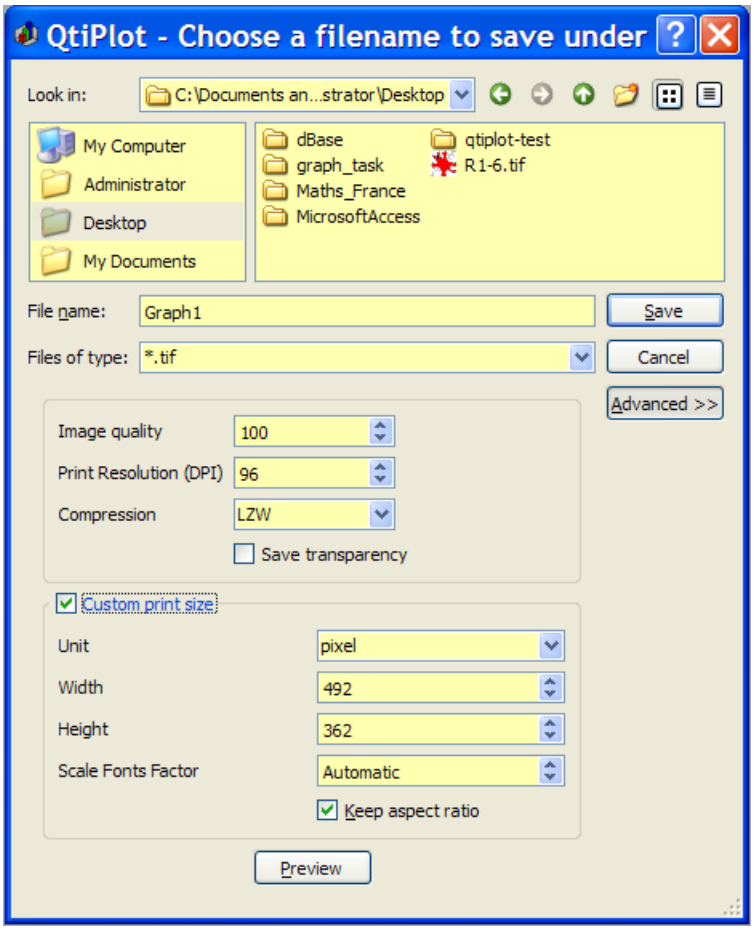
3.1.16 File-> Print All Plots

Prints all plots in the project. A print dialog is opened where you can select the printer, different paper sizes, etc.

3.1.17 File -> Export Graph

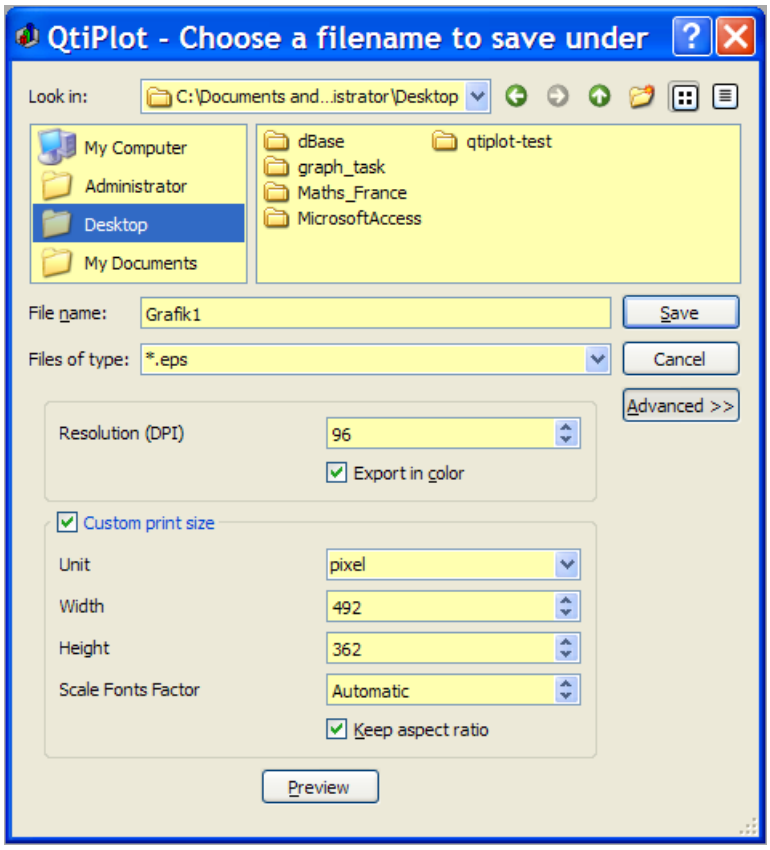
The **Export Graph** command appears in the file menu whenever a graph window is selected. All graphs, or one graph at a time, can be exported in any of the available image formats. Since all export options have the same image formats available, these will be described first. Depending upon the image format chosen, you may be able to customize some image file parameters by checking the *show options* check box. Available options vary according to the format chosen.

For the *bmp, pbm, jpeg, xbm, pgm, ppm* image formats, the only available option is the quality of the image. This parameter defines the image compression ratio, and may be set to any value between 0 and 100%. Higher values produce a better quality image and a larger file, while lower values result in increasingly lossy compression, degraded image quality, and smaller file size. For *png*, *tiff* and *xpm*, there is an option to choose a transparent background.

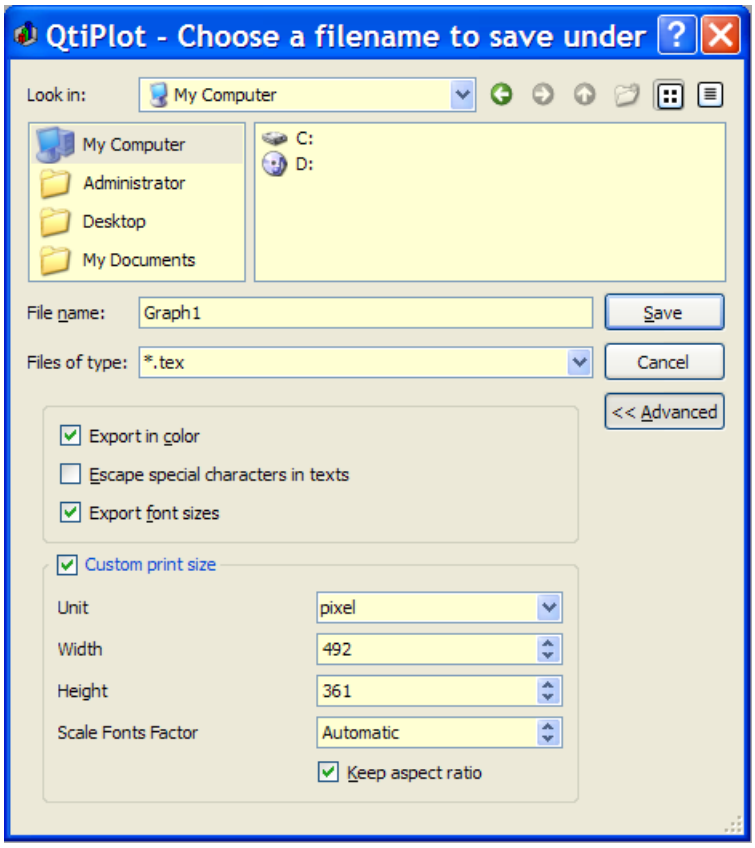


For *eps*, *pdf*, and *ps* file formats, the option dialog is different again. The main parameters available are the desired resolution and the size of the sheet of paper for which the image will be formatted. The default value for the resolution is the screen resolution. If you increase the resolution, the overall size of the plot will be unchanged but the quality of the graphic elements will be better.

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



When exporting the plot to LaTeX (.tex) there are two very useful options: *Export font sizes* and *Escape special characters in title/axis labels*. If checked, the first option will include LaTeX commands in the output which keep the original font sizes. If not checked, 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 whether 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 must uncheck this option.



3.1.17.1 Export Graph -> Current (Alt-G)

This selection will save the active graph using one of the image formats described above.

3.1.17.2 Export Graph -> All (Alt-X)

This selection will save all graphs in the project using one of the image formats described above.

3.1.17.3 File -> Create Open Document Presentation...

This selection will save all the graphs in the project in an Open Document Format file (.odf) that can be opened and edited with OpenOffice.

3.1.18 File -> Export

The **Export** command appears in the file menu whenever a non-graph window (i.e., Table or Matrix) is selected.

3.1.18.1 Export ASCII

This command opens the Export ASCII dialog, with which you can save the active table or matrix in a chosen ASCII format (".dat", ".html", ".odf", ".tex", ".txt", and ".xls" are available).

3.1.18.2 Export Excel

This command opens the Export ASCII dialog with the ".xls" format pre-selected. This saves the active table or matrix as an Excel spreadsheet.

3.1.18.3 Export to PDF (Ctrl-Alt-P)

This command opens a generic file dialog to save the active window as a PDF document.

3.1.19 File -> Import

3.1.19.1 File -> Import -> Import ASCII... (Ctrl-K)

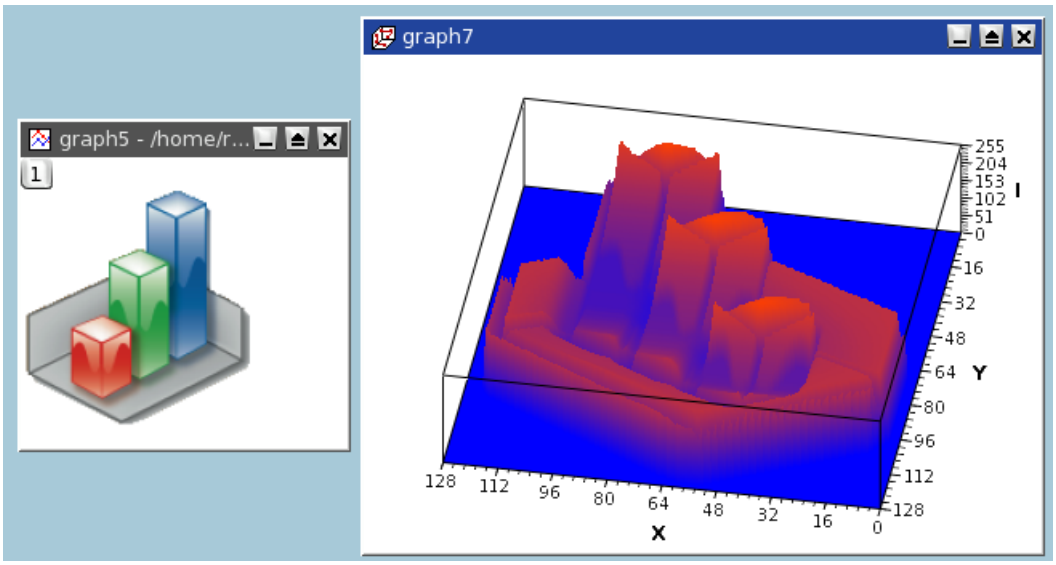
Opens the Import dialog used to import ASCII data files. The file to import, and the options for importation are set in this dialog.

3.1.19.2 File -> Import -> Sound (WAV)...

This option allows you to import an uncompressed sound (.wav) file (PCM format).

3.1.19.3 File-> Import Image...

Using this command, an image may be loaded into a QtiPlot project and converted into 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 drawn from the intensity matrix obtained from the QtiPlot logo.

3.1.20 File -> Quit (Ctrl-Q)

Closes the application. You will be asked whether or not you want to save any changes.

3.2 The Edit Menu

3.2.1 Edit -> Undo (Ctrl-Z)

Undo the last change to a note or matrix. Currently this function does not work for anything other than notes and matrices.

3.2.2 Edit -> Redo (Ctrl-Shift-Z)

Reverse the effect of the last "Undo" operation on a note or a matrix. Currently this function does not work for anything other than notes and matrices.

3.2.3 Edit -> Cut Selection (Ctrl-X)

While this command does not currently appear in the Edit Menu, the functionality is provided with the Ctrl-X key. The command copies the current selection to the clipboard and deletes the selection. It currently works for tables and for 2D plots objects.

3.2.4 Edit -> Copy Selection (Ctrl-C)

Copies the current selection to the clipboard. It currently works only for layers.

3.2.5 Edit -> Paste Selection (Ctrl-V)

Pastes the content of the clipboard to the active window. It currently works only for layers.

3.2.6 Edit -> Delete Selection (Del)

Removes the current selection from the project. It currently works only for layers.

3.2.7 Edit -> Delete Fit Tables

Each time you fit your data to some mathematical model, a new table is created in which 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 completed several tentative fits, a number of fit result tables may be present in your project. This command allows you to remove the results of all the different fits that you have tested.

3.2.8 Edit -> Clear Log Information

Allows the user to clear the log panel. All history information about analyses performed by the user will be deleted from the project file. The log panel will then be empty.

3.2.9 Edit -> Preferences...

Opens the Preferences dialog.

3.3 The View Menu

3.3.1 View -> Toolbars... (Ctrl-Shift-T)

Opens a pop-up menu allowing you to enable/disable tool bars.

3.3.2 View -> Plot Wizard (Ctrl-Alt-W)

Opens the Plot Wizard dialog.

3.3.3 View -> Project Explorer (Ctrl-E)

Opens/Closes the Project Explorer dialog, 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.

3.3.4 View -> Results log

Opens/Closes a panel displaying a history of all data analysis operations performed by the user.

3.3.5 View -> Undo/Redo Stack...

Shows/Hides the matrix Undo/Redo Stack window. The Undo/Redo stack contains the history of editing changes for a selected matrix. When any matrix is selected, its Undo/Redo information will be copied into this window. If another matrix is selected, the data for that matrix will be swapped into the list. Only one matrix can have it's Undo/Redo information showing at a time. The depth of the stack (i.e., number of possible undo steps) is set in the "Applications" tab in the "General" section of the Preferences dialog. Each list entry consists of the name of the matrix followed by the identity of the cell that was edited. Each edit pushes another entry onto the list. The current position in the Undo/Redo history is highlighted. Undo (Ctrl-Z or) or Redo (Ctrl-Shift-Z or) will move the highlight as appropriate. The history can be rapidly traversed by clicking on a stack entry, which will move the highlight to the clicked entry and revert the matrix to that point in the history. Each time the project is saved, the save-project icon ( ) is placed next to the stack entry that was highlighted at the time of the file save operation. This serves as a reminder to the user of exactly what version of the matrix resides on disk.

3.3.6 View -> Show/Hide Scripting Console

Shows/Hides the scripting console window.

3.4 The Scripting Menu

3.4.1 General Scripting Commands

These are commands that always appear in the Scripting menu, regardless of the window type selected

3.4.1.1 Scripting -> Scripting language

Opens a dialog used to select the scripting language for the current project.

3.4.1.2 Scripting -> Restart scripting

Reinitializes the scripting environment.

3.4.1.3 Scripting -> Add Custom Script Action...

Opens the Custom Action dialog, which allows you to define menu items and toolbar buttons that launch Python scripts.

3.4.2 Notes Specific Scripting Commands

If the active window in the project is a Notes window, the following additional items will also be available in the scripting menu:

3.4.2.1 Scripting -> Execute (Ctrl+J)

Executes the line where the mouse cursor is placed in the Notes window.

3.4.2.2 Scripting -> Preferences... (Ctrl+Shift+J)

Executes all lines in the Notes window.

3.4.2.3 Scripting -> Evaluate (Ctrl+Return)

Evaluates the line where the mouse cursor is placed in the Notes window.

3.4.2.4 Rename Tab...

Opens a standard dialog which permits changing the name of the active notes tab.

3.4.2.5 Add Tab

Adds a new tab to the active notes window. The new tab will be named *untitled* by default. The tab may be renamed at any time using the Rename Tab... command.

3.4.2.6 Close Tab

Closes (deletes) the active Notes tab. The contents of the tab will be lost. No warning is given!

3.5 The Graph Menu

This menu is only active (visible) when a graph window is selected.

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

Opens the Add/Remove Curves... dialog, allowing easy addition or removal of curves from the active plot layer. This dialog can also be used to modify a curve which is already plotted by changing the columns which are used as X or Y values. The curve is added to the currently active layer. If there is no layer on the graph, an error message will pop-up.

3.5.2 Graph -> Add Function... (Ctrl-Alt-F)

Opens the Add Function... dialog. This command is used to add new function curves to an existing layer. New curves are added to the currently active layer.

3.5.3 Graph -> Add Error Bars... (Ctrl-B)

Opens the Add Error Bars... dialog. You can add error bars to X and/or Y values on an existing plot. Error bars are added to the currently active layer.

3.5.4 Graph -> New Legend (Ctrl-L)

Adds a new legend object to the active layer. You can have more than one legend on a plot. These legends can later be customized by double clicking on a given legend.

3.5.5 Graph -> Add Equation (Alt-Q)

This command is used to add a *Tex* formatted equation on a layer. When selected, the cursor changes to an edit text cursor. You must then click in the plot area to specify the position of the new Tex equation. A Tex equation editor will pop-up allowing you to enter the equation to be displayed and set all its properties (color, font, etc...)

3.5.6 Graph -> Add Text (Alt-T)

This command is used to add text items on a layer. When selected, the cursor changes to an edit text cursor. You must then click in the plot area to specify the position of the new text box. A text dialog will pop-up allowing you to type the text to be displayed and set all its properties (color, font, etc...)

3.5.7 Graph -> Draw Arrow (Ctrl-Alt-A)

Changes the active layer's operational mode to 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. Switch back to the normal operating mode by clicking the "Pointer" icon in the Plot toolbar.

3.5.8 Graph -> Draw Line (Ctrl-Alt-L)

Changes the active layer's operational mode to drawing mode. You must click on the layer canvas in order to specify the starting point for the new line, and then click once more to specify its ending point. You can edit the new line using the line dialog. Switch back to normal operating mode by clicking the "Pointer" icon in the Plot toolbar.

3.5.9 Graph -> Add Rectangle (Ctrl-Alt-R)

Draws a rectangular box on the active plot layer. After selecting this command, the pointer changes to a crosshair target. Click and hold the left mouse button on the active layer to indicate the first corner of the rectangle, then drag the cursor to the opposite corner of the rectangle. Once the second point is specified, release the left mouse button to draw the rectangle. The pointer automatically changes back to normal mode.

3.5.10 Graph -> Add Ellipse (Ctrl-Alt-E)

Draws a elliptical shape on the active plot layer. After selecting this command, the pointer changes to a crosshair target. Click and hold the left mouse button on the active layer to indicate the first corner of the ellipse's bounding rectangle, then drag the cursor to the opposite corner of the bounding rectangle. Once the second point is specified, release the left mouse button to draw the ellipse. The pointer automatically changes back to normal mode. Circles may be drawn using this command by setting the height and width equal. (The object properties dialog can be used to set height and width to exactly the same value.)

3.5.11 Graph -> Add Time Stamp (Ctrl-Alt-T)

This command is used to add a special label in the active layer which contains the current date and time. The properties of this label can be customized like any other label added by the Add Text.

The date and time copied into a timestamp label is not modified later if the plot is modified, saved, etc.

3.5.12 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 with the mouse using drag-and-drop.

3.5.13 Z-Order Commands...

The zorder commands act upon the last few items. Even though they do not appear in the graph menu, they are in the plot toolbar and in the object drop down lists when right clicking a selected object.

3.5.13.1 Move to Top

Although this command does *not* appear in the Graph menu, it does act upon the objects created using the last few commands (specifically legend, equation, text, timestamp, ellipse, rectangle and image objects). When a suitable object is selected, this command will be available, either from the Plot toolbar or from the drop-down list that appears when right-clicking on a selected object. When the command is used, the object will be made visible by being promoted to the front of the plot layer. It does not affect the z-order relative to other objects. Their relative z-order depends only upon their order of creation.

3.5.13.2 Move to Bottom

Although this command does *not* appear in the Graph menu, it does act upon the objects created using the last few commands (specifically legend, equation, text, timestamp, ellipse, rectangle and image objects). When a suitable object is selected, this command will be available, either from the Plot toolbar or from the drop-down list that appears when right-clicking on a selected object. When the command is used, the object will be made invisible by being demoted to the back of the plot layer. It does not affect the z-order relative to other objects. Their relative z-order depends only upon their order of creation.

3.5.14 Graph -> Add Layer (Alt-L)

Adds a new layer to a graph window. The command opens a dialog which allows 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).

3.5.15 Graph -> Add Empty Inset Layer

This is a convenience command that creates a new, small sized, empty layer located on top of the currently selected layer. Once this layer is created, you can further resize, move, set it's properties and add curves to it as needed. Inset layers are not linked to another layer in any way. They just happen to be located in the same place on the graph window. In fact the inset layer can be moved anywhere on the graph window and is treated like a normal layer in every way. When the inset layer is completely contained inside another layer, it will disappear under that layer when it is selected. The new inset layer has it's own layer select button in the upper left of the graph window. Clicking on this button will bring the inset back to the front, rendering it visible again.

3.5.16 Graph -> Add Inset Layer With Curves

This command is essentially identical to the **Add Empty Inset Layer** command, with the single exception that any curves contained in the currently active layer are duplicated in the new inset layer.

3.5.17 Graph -> Arrange Layers (Shift-A)

Opens the Arrange layers dialog. This dialog allows you to customize the layout of the layers in the active graph window.

3.5.18 Graph -> Automatic Layout command

Automatically arranges all the layers in the active graph window. There must be at least 2 layers on the graph. Layers are packed into the smallest dimensioned array possible. All layers are scaled to fit into the existing graph window. Rows are added until a the smallest square array is filled, at which point an additional column is added. Rows are then filled and additional rows added until the array is again square. For example, 2 layers results in a 1x2 array. 3 layers produces a 2x2 array with an empty space in the second row. 4 layers produces a full 2x2 array. Adding another layer will produce a 2x3 array, with one empty space in the second row. Additional layers result in more rows and columns being added to contain all the layers.

3.5.19 Graph -> Extract to Graphs command

Each layer in the currently active graph window is copied into its own, newly created, graph window. The scale and size of the new layer is the same as the original layer. The original graph window remains unchanged.

3.5.20 Graph -> Extract to Layers command

Each curve in the currently active layer is moved onto a newly created layer. The original layer is destroyed. The newly created layers are then auto arranged (see the Automatic Layout command).

3.5.21 Graph -> Remove Layer (Alt-R)

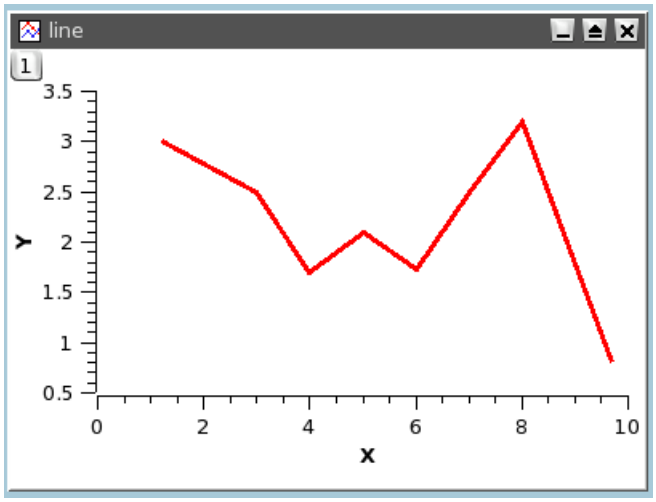
Deletes the active layer and pops-up a question dialog allowing you to choose whether the remaining layers should be automatically re-arranged or not.

3.6 The Plot Menu

This menu is active only when a table is selected. These commands allow plotting data from the selected table. As a group, all of these commands create a new graph window which contains a single, empty layer. The newly created plot is drawn on this empty layer.

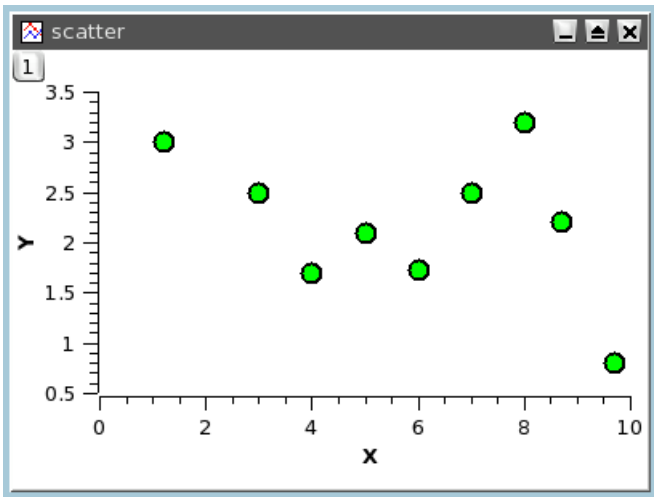
3.6.1 Line

Plots the selected data columns using "Line" style. The command can also be activated by clicking on the ⟋ icon of the Table toolbar. Once the plot is created, the drawing style of the data series can be customized with the Custom curves dialog.



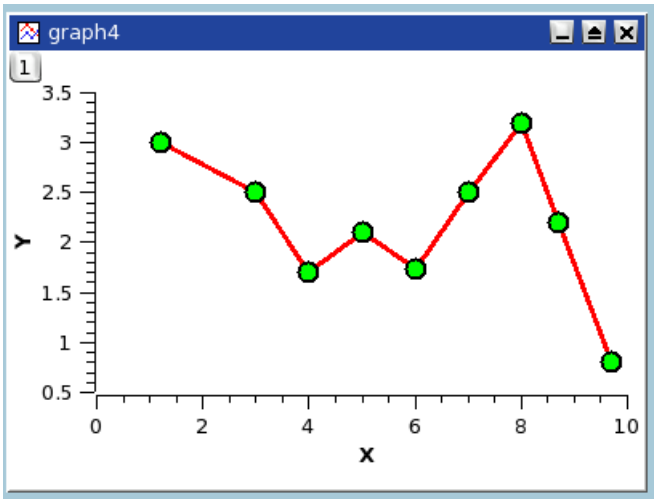
3.6.2 Scatter

Similar to the **Line** command except that the plot is drawn using "Scatter" style. The command can also be activated by clicking on the ⋰ icon of the Table toolbar. Once the plot is created, the drawing style of the data series can be customized with the Custom curves dialog.



3.6.3 Line+Symbol

Identical to the **Line** command except that the plot is drawn using "Line + Symbol" style. This command can also be activated by clicking on the icon of the Table toolbar. Once the plot is created, the drawing style of the data series can be customized with the Custom curves dialog.

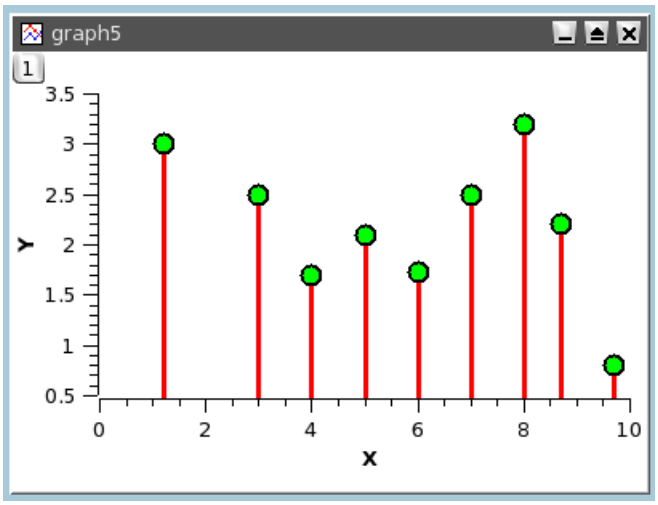


3.6.4 Special Line+Symbol ->

Selecting **Special Line+Symbol** opens up a sub-menu of additional commands for plotting specialized graphs consisting of lines and symbols.

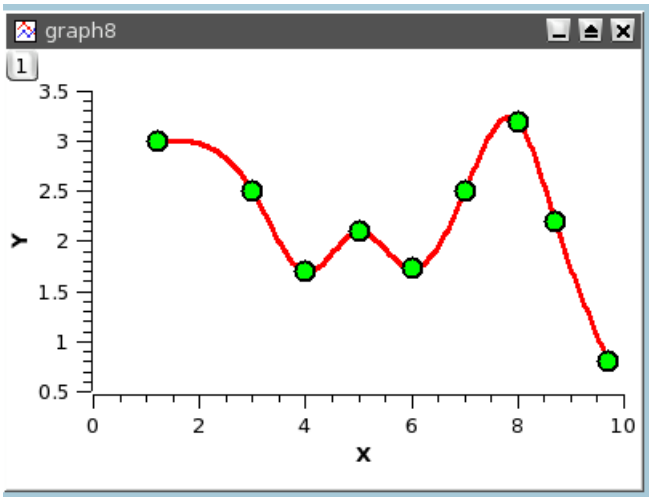
3.6.4.1 Vertical Drop Lines

Plots the selected data columns using "Vertical drop lines" style. Once the plot is created, the drawing style of the data series can be customized with the Custom curves dialog.



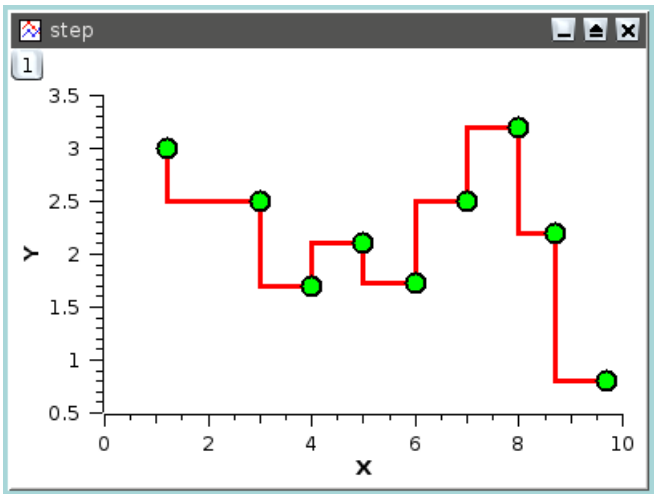
3.6.4.2 Spline

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



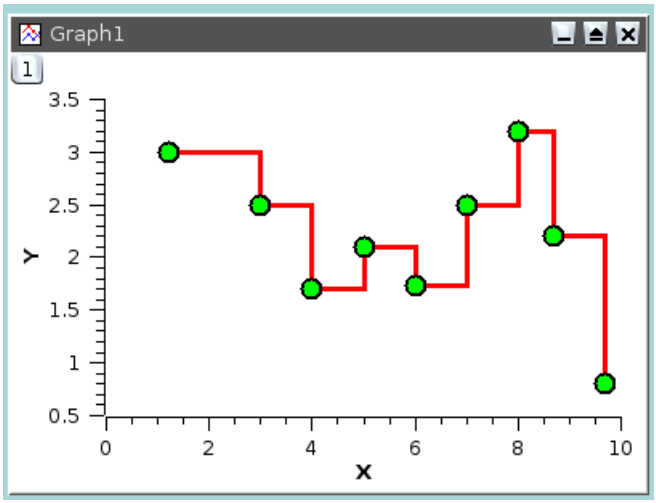
3.6.4.3 Vertical Steps

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



3.6.4.4 Horizontal Steps

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

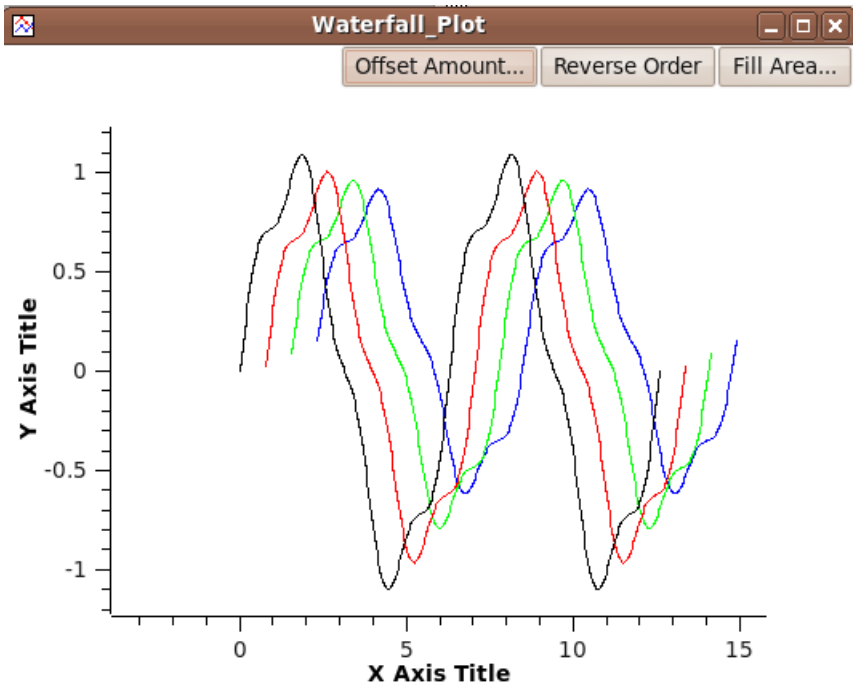


3.6.4.5 Double-Y

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

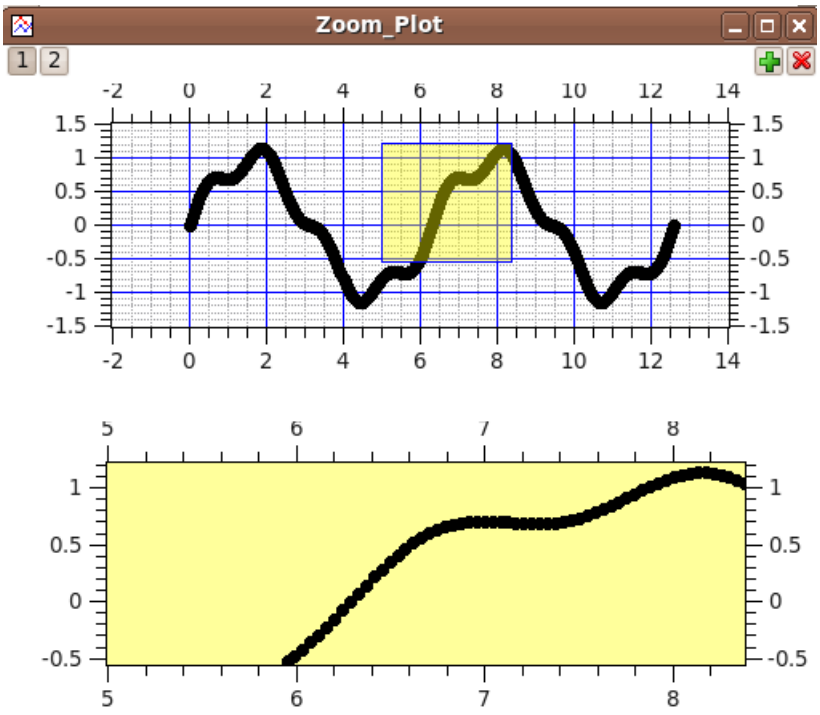
3.6.4.6 Waterfall

Creates a new graph window containing a waterfall plot generated from data in the selected columns. Each column is drawn as a separate curve which is offset from the previous one in both *X* and *Y*, creating the so-called waterfall effect. The figure shown below is a simple example of a waterfall plot where each column is simply an attenuated version of the previous column.



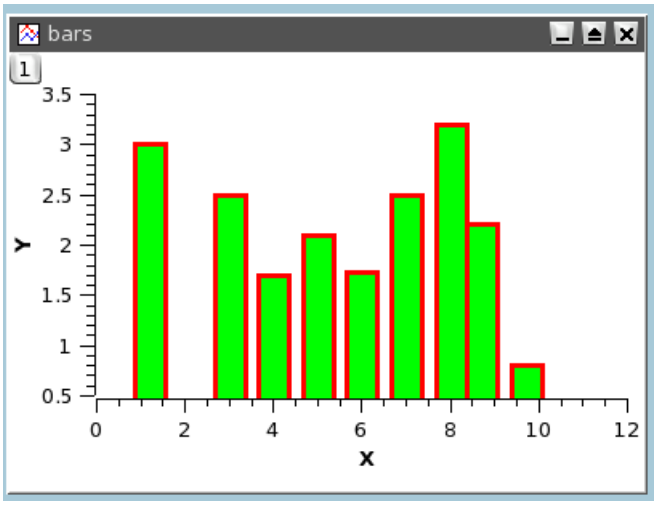
3.6.4.7 Zoom

Creates a new graph with 2 layers. The first is a standard plot layer which contains a curve for each selected *Y* column. The second is a special layer which has 2 components: 1) There is a standard plot layer that shows a portion of the first layer's surface in a magnified view; 2) There is a separate graphical window that can be sized and dragged around the first layer to specify the portion of that layer which is shown in the magnified view. The figure below shows a typical *zoom* plot.



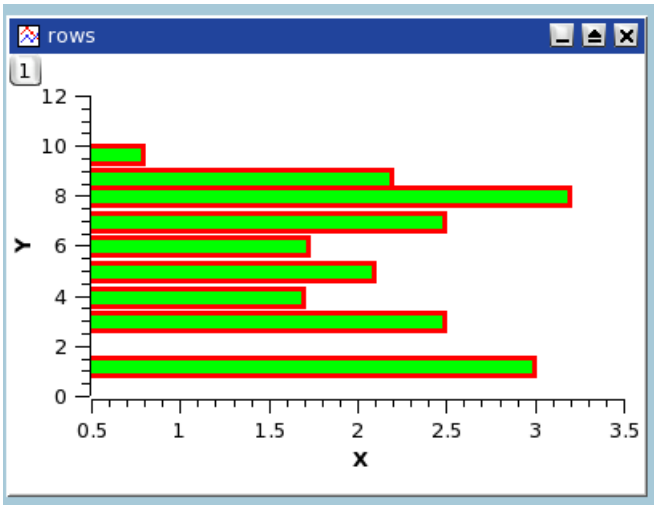
3.6.5 Columns

Plots the selected data columns using "Columns" style, that is vertical bars.



3.6.6 Rows

Plots the selected data columns using "Rows" style.

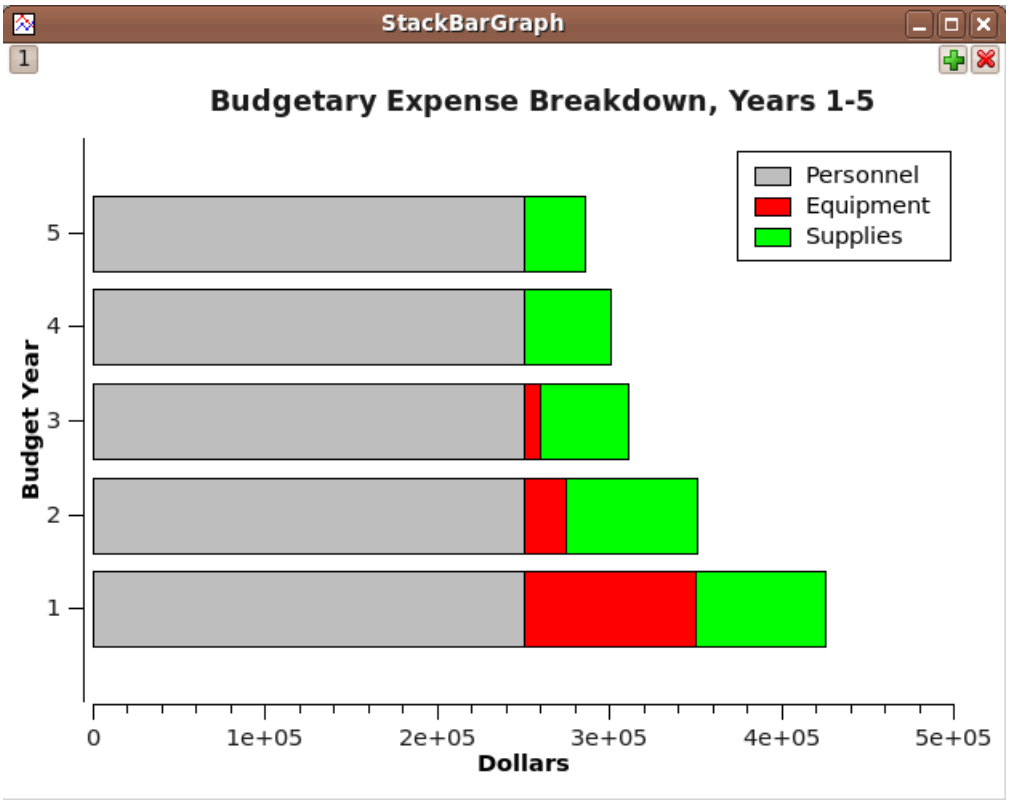


3.6.7 Special Bar/Column ->

Selecting **Special Bar+Column** opens up a sub-menu of commands for plotting specialized bar and column graphs.

3.6.7.1 Stack Bar

The **Stack Bar** command plots a stack-bar for each selected row in the currently active table. They may be used to conveniently display cumulative data that changes over time, such as budgetary information. The stack-bars are drawn one above the other in a position determined by the X value of the corresponding row. Each stack-bar is composed of a set of joined segments. The widths of the segments are set by the Y values in the row and their relative positions correspond to the column order in the table. Each segment is plotted in a color corresponding to the column number of the Y value determining the segment's width. Colors, segment outlining, vertical separation of the stack-bars, and the axis values can all be customized. An example of a stack-bar plot is shown below, followed by the table that was used to generate the plot.



A stack-bar plot showing a hypothetical budgetary breakdown for a 5 year period. The data in the table shown below was used to generate this plot.

StackBarData

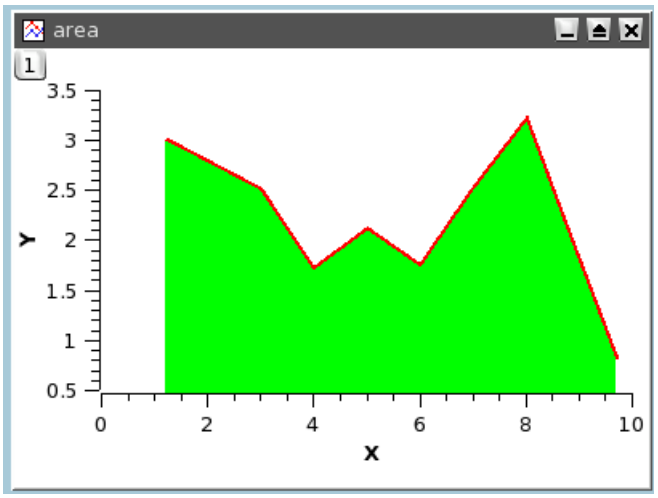
	Year[X]	Personnel[Y]	Equipment[Y]	**Supplies[Y]**
1	1	250,000	100,000	75,000
2	2	250,000	25,000	75,000
3	3	250,000	10,000	50,000
4	4	250,000	0	50,000
5	5	250,000	0	35,000
6				

3.6.7.2 Stack Column

The **Stack Column** command plots a stack-column for each selected row in the currently active table. It is essentially the same as the Stack Bar plot, but drawn as vertical stack-columns rather than horizontal stack-bars.

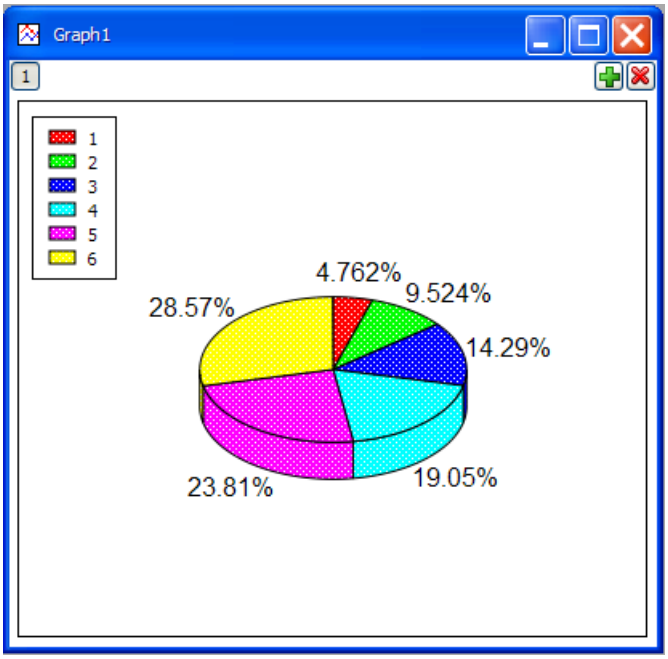
3.6.8 Area

Plots the selected data columns using "Area" style.



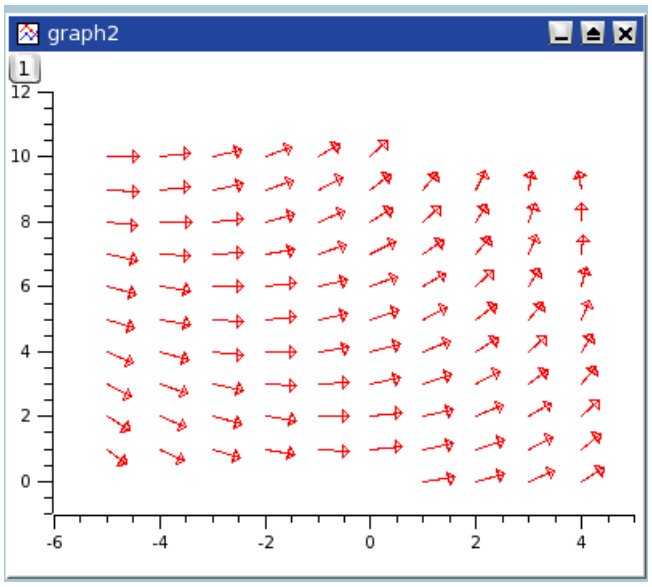
3.6.9 Pie

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



3.6.10 Vectors XYXY

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



3.6.11 Vectors XYAM

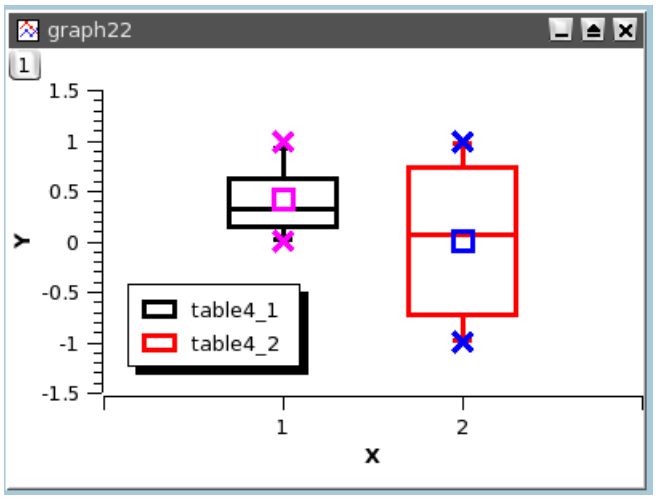
Creates a vectors plot using the selected columns in the active table window. You must select four columns for this particular type of plot. The first two columns give the coordinates of the starting points of the vectors, the last two columns give the angles (in radians) and magnitudes of the vectors.

3.6.12 Statistical Graphs ->

Selecting the *Statistical Graphs* -> item opens up a sub-menu of commands for plotting various statistical graphs. Statistical plots do not give a direct drawing of the selected table data. However, a representation of the frequency distribution of the Y-values is plotted.

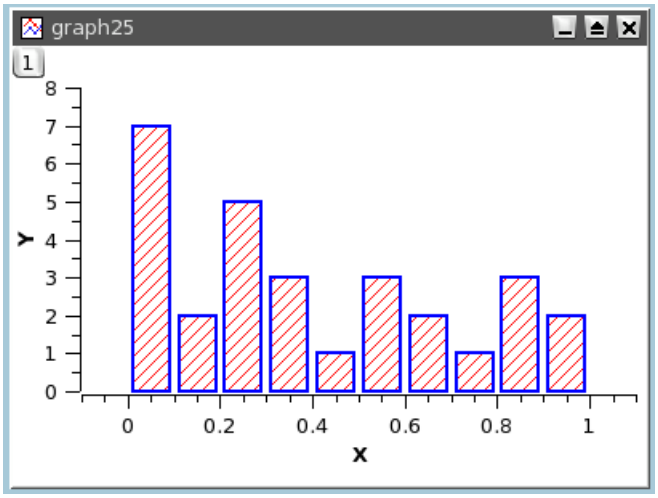
3.6.12.1 Box Plot

Creates a box plot using the selected data columns in the active table window. This type of plot is used to give a graphical representation of some of the classical parameters of a frequency distribution, such as the mean, 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.



3.6.12.2 Histogram

Creates a frequency histogram 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.



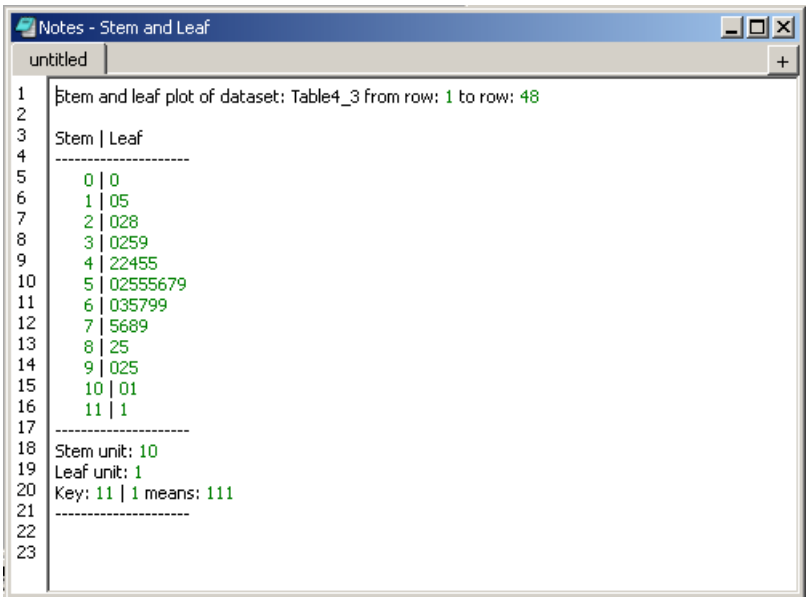
Note that this command plots a frequency distribution that is *computed* from your data. If you want to draw a histogram directly from data values, use the **Bars** plot instead.

3.6.12.3 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 Vertical 2 Layers command for more details.

3.6.12.4 Stem and Leaf

Generates a stem-and-leaf tabular plot from the selected column in a new *note* window. Stem-and-Leaf plots are similar to histograms but are represented using rows of decimal digits. Like a histogram, data is binned, usually by taking the most significant digits (MSDs) as "stem" values which are listed in a column on the left. The least significant digit(s) from all values having identical MSDs are then listed to the right of each stem value. From a statistical point of view, this gives the appearance of a histogram but preserves the actual data values in the plot. When this command is executed, you will be given the opportunity to select/confirm the binning increment, which is given as a power of 10 and defaults to a power of 1 (that is, a binning increment of 10).



This figure shows a typical stem-and-leaf representation of 48 random data points which are roughly Gaussian (i.e., normally distributed) and uses the default binning increment of 10.

3.6.13 Panel ->

Selecting the *Plot -> Panel ->* menu item opens a sub-menu of commands that can be used to quickly obtain some commonly used arrangements of multiple plots. Unlike other commands in this menu, the new graph window that is created may have more than one layer (as needed).

3.6.13.1 Vertical 2 Layers

Creates 2 vertically stacked layers. The data columns selected in the active table window (one curve per layer) are plotted on these layers.

3.6.13.2 Horizontal 2 Layers

Creates 2 horizontally stacked layers. The data columns selected in the active table window (one curve per layer) are plotted on these layers.

3.6.13.3 4 Layers

Creates 4 layers on a 2x2 grid. The data columns selected in the active table window (one curve per layer) are plotted on these layers.

3.6.13.4 Stacked Layers

Creates a group of vertically stacked layers, one layer per Y-Coordinate column selected. Data columns selected in the active table window (one curve per layer) are plotted on these layers.

3.6.13.5 Custom Layout...

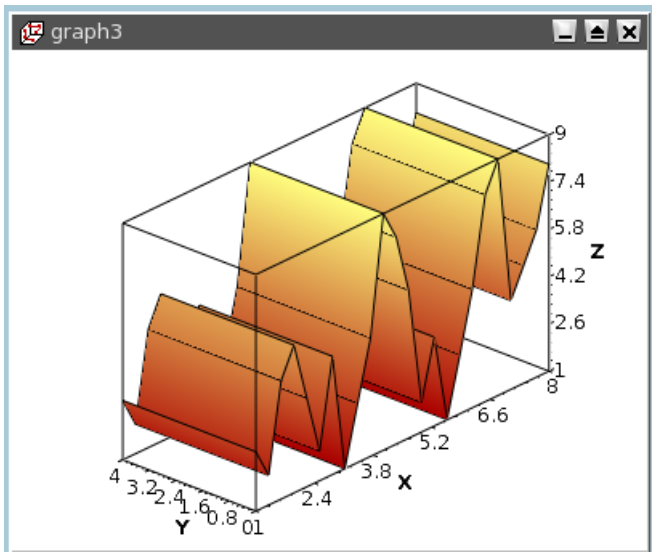
Opens the *Arrange Layers* dialog which allows complete customization of the layer layout.

3.6.14 Data -> Plot 3D ->

Selecting the *Plot -> Plot 3D ->* menu item opens a sub-menu of commands used to draw some common 3D plots of 2D data.

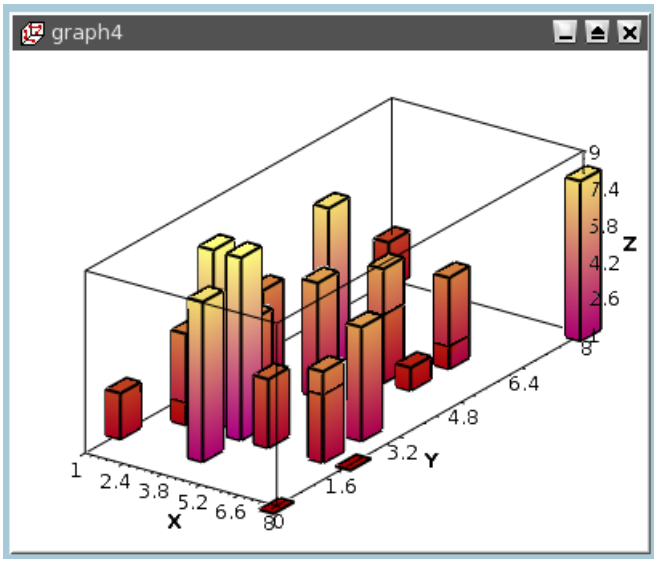
3.6.14.1 Ribbons

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



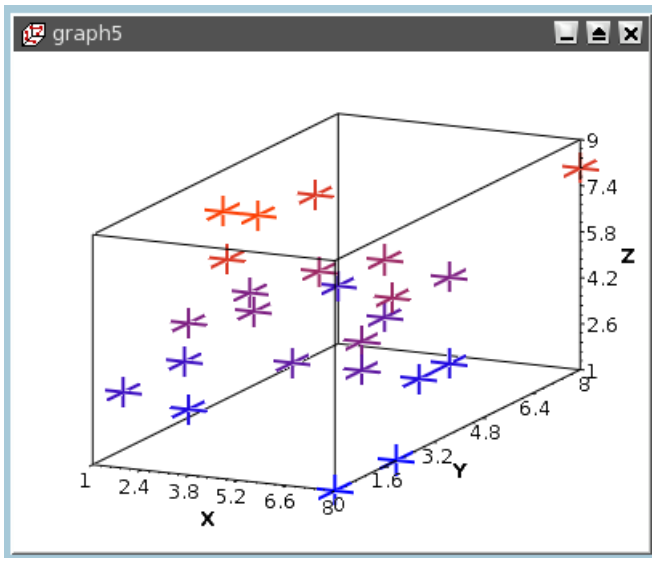
3.6.14.2 Bars

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



3.6.14.3 Scatter

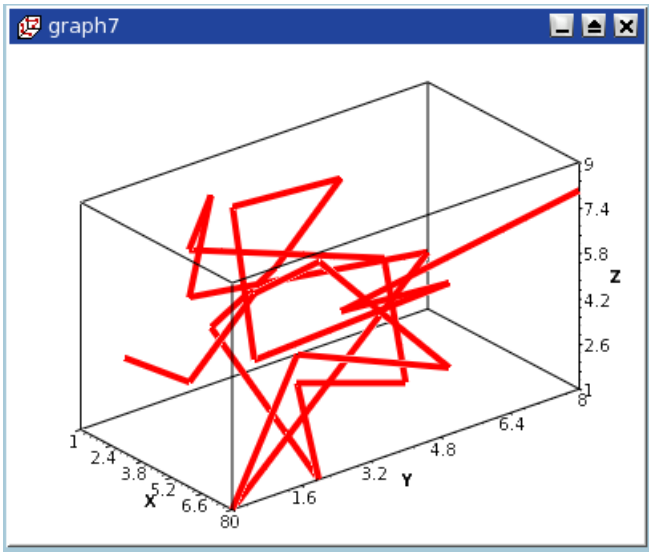
Makes a 3D plot of the selected data column in the active table window (only one column allowed) using "3D Dots" style. The 3D point symbol style can be changed via the 3D Plots Settings dialog.



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.

3.6.14.4 Trajectory

Makes a 3D plot of the selected data column in the active table window (only one column allowed) using "3D Line" style. The line width and color may be changed via the 3D Plots Settings dialog.

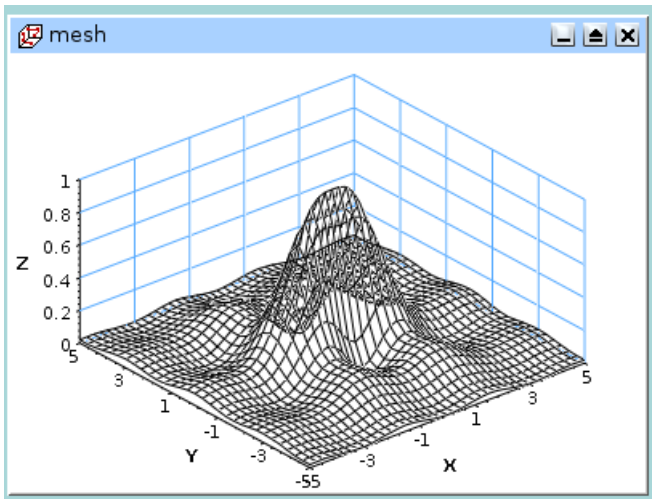


3.7 The 3D Plot menu

This menu is only active (visible) when a matrix is selected.

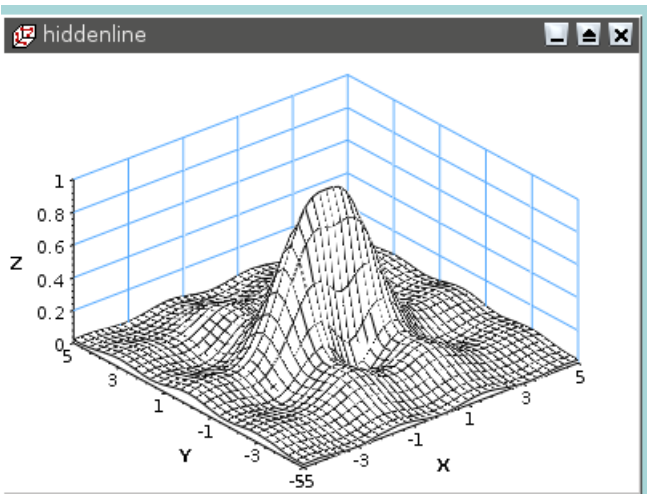
3.7.1 3D Wire Frame

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



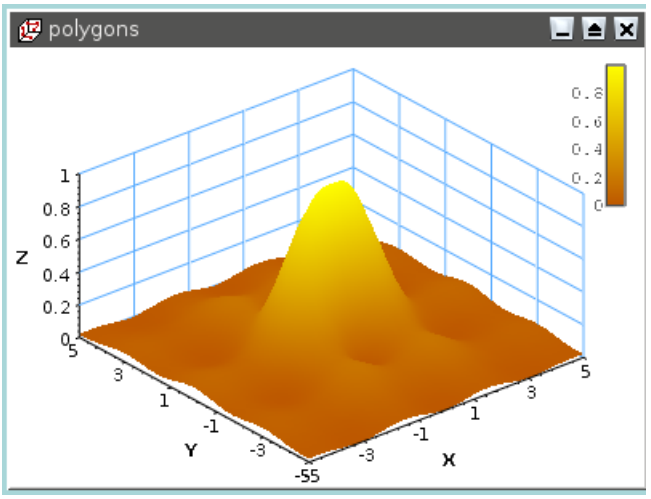
3.7.2 3D Hidden Lines

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



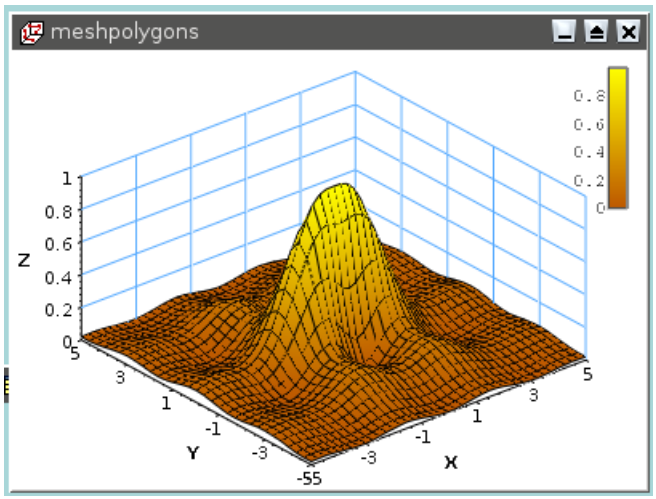
3.7.3 3D Polygons

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



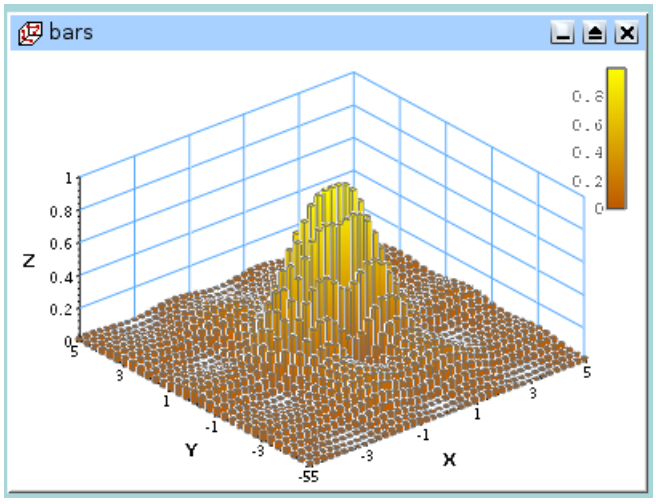
3.7.4 3D Wire Surface

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



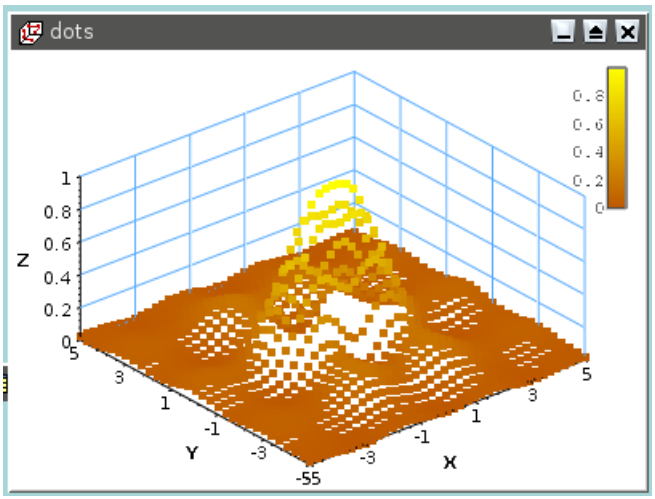
3.7.5 Bars

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



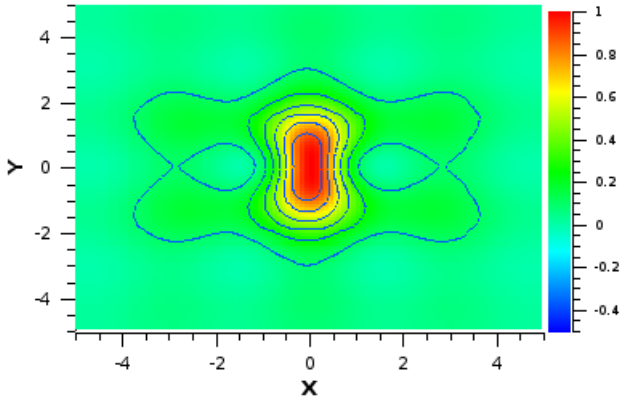
3.7.6 Scatter

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.



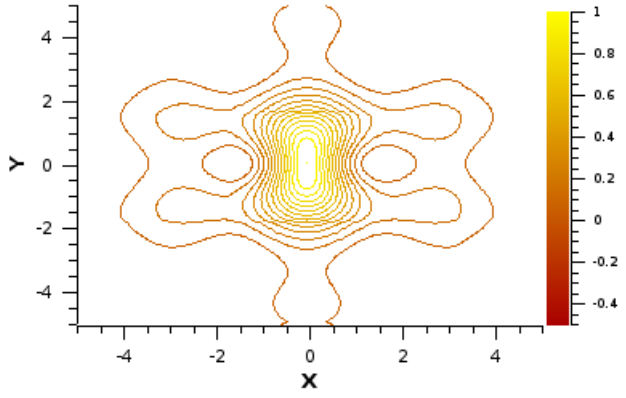
3.7.7 Contour+Color Fill

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 activate the Contour Options Dialog.



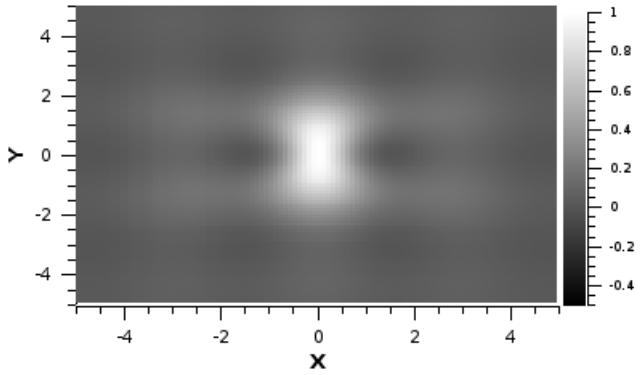
3.7.8 Countour Lines

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 activate the Contour Options Dialog.



3.7.9 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 activate the Contour Options Dialog.



3.8 The Data Menu

This menu is visible and active only when a graph window is selected. Commands in this menu will act on the plot/layer that is currently active in the graph window. All of these commands are switches - that is, they switch on an operational mode which remains in effect until it is canceled by selecting another command or by using the **Disable tools** command.

3.8.1 Data -> Disable tools

For many of the remaining commands in this menu, QtiPlot will be locked into a special mode corresponding to that command, and that command's special pointer will be used instead of the normal pointer. This command is used to exit any of these special modes, returning QtiPlot to its normal mode and pointer behavior.

3.8.2 Data -> Zoom In/Out and Drag Canvas

Switches the active plot layer into a dynamic zoom/drag mode. Upon command execution, the default mode is "dynamic drag". To drag the plot, simply left click anywhere in the plotting area and drag the plot around with the mouse. Drop it at the new location by releasing the left mouse button. To zoom the plot, switch to dynamic zoom mode by first right clicking somewhere in the plot area. (This will probably pop-up a context menu that you should ignore.) Next, left click at a suitable zoom reference point in the plot area to enter dynamic zoom mode (this will also dismiss any pop-up menu). Once in dynamic zoom mode, scale the plot by moving the mouse cursor up to zoom out, or down to zoom in. Right and left mouse movements have no effect. You should use some foresight in picking the zoom reference point. For example, selecting a point at the bottom of the plot area will limit zooming to upward mouse travel - that is, you will only be able to zoom out. Once a suitable zoom factor has been found, left click anywhere in the plot to accept the zoom and return to dynamic drag mode.

Note that the mouse wheel is always active when using this command. It can be used to zoom while in either dynamic drag or dynamic zoom modes. Also note that in either mode, zooming leaves the plot's aspect ratio unchanged.

3.8.3 Data -> Zoom/Drag Canvas Horizontally

Switches the active plot layer into a dynamic zoom/drag mode which is limited to the horizontal direction. Upon command execution, the default mode is "dynamic horizontal drag". To drag the plot, simply left click anywhere in the plotting area and drag the plot to the right or left with the mouse. Drop it at the new location by releasing the left mouse button. Note that it may appear that you can move the plot up or down while dragging, but the original vertical location will be restored when the plot is dropped at the new horizontal location. To zoom the plot, switch to dynamic horizontal zoom mode by first right clicking

somewhere in the plot area. (This will probably pop-up a context menu that you should ignore.) Next, left click at a suitable zoom reference point in the plot area to enter dynamic horizontal zoom mode (this will also dismiss any pop-up menu). Once in dynamic horizontal zoom mode, scale the plot's width by moving the mouse cursor up to zoom out, or down to zoom in. Right and left mouse movements have no effect. You should use some foresight in picking the zoom reference point. For example, selecting a point at the bottom of the plot area will limit zooming to upward mouse travel - that is, you will only be able to zoom out. Once a suitable zoom factor has been found, left click anywhere in the plot to accept the zoom and return to dynamic horizontal drag mode.

Note that the mouse wheel is always active when using this command. It can be used to zoom while in either dynamic horizontal drag or dynamic horizontal zoom modes. Also note that in either horizontal mode, zooming will change the plot's aspect ratio.

3.8.4 Data -> Zoom/Drag Canvas Vertically

Switches the active plot layer into a dynamic zoom/drag mode which is limited to the vertical direction. Upon command execution, the default mode is "dynamic vertical drag". To drag the plot, simply left click anywhere in the plotting area and drag the plot up or down with the mouse. Drop it at the new location by releasing the left mouse button. Note that it may appear that you can move the plot right or left while dragging, but the original horizontal location will be restored when the plot is dropped at the new vertical location. To zoom the plot, switch to dynamic vertical zoom mode by first right clicking somewhere in the plot area. (This will probably pop-up a context menu that you should ignore.) Next, left click at a suitable zoom reference point in the plot area to enter dynamic vertical zoom mode (this will also dismiss any pop-up menu). Once in dynamic vertical zoom mode, scale the plot's height by moving the mouse cursor up to zoom out, or down to zoom in. Right and left mouse movements have no effect. You should use some foresight in picking the zoom reference point. For example, selecting a point at the bottom of the plot area will limit zooming to upward mouse travel - that is, you will only be able to zoom out. Once a suitable zoom factor has been found, left click anywhere in the plot to accept the zoom and return to dynamic vertical drag mode.

Note that the mouse wheel is always active when using this command. It can be used to zoom while in either dynamic vertical drag or dynamic vertical zoom modes. Also note that in either vertical mode, zooming will change the plot's aspect ratio.

3.8.5 Data -> Zoom in (Ctrl-+)

Switches the active plot layer to zoom mode. The mouse cursor shape changes to a magnifying lens when it is inside a plot canvas. To zoom the plot, draw a box around a selected part of the plot by clicking at one corner of the desired box, dragging the mouse pointer to the opposite corner of the box and clicking a second time. The plot will be zoomed up such that the contents of the drawn box now fills the existing plotting area.

3.8.6 Data -> Zoom out (Ctrl--)

This command cancels a previous zoom operation. A zoom history is kept so that you can do multiple zoom out commands to step backwards through the zoom history. A separate zoom history is maintained for each plot layer. Executing this command does not set any special mode of its own but it will cancel any other special mode that is currently in effect (i.e., it performs an implicit **Disable tools** command).

3.8.7 Data -> Rescale To Show All (Ctrl-Shift-R)

Rescales the active plot layer after a zoom operation so that all data on the graph is visible. This command does not set any special mode of its own, but unlike the **Zoom out** command, it does not alter any other mode that may be in effect. Nevertheless, it does delete the zoom history of the rescaled plot.

3.8.8 Data -> Data Reader (Ctrl-D)

Shows a red cross cursor and opens the Data Display toolbar giving easy and quick access to the values of the data points. Select data points by moving the cursor with the Left and Right arrow keys or, more easily and quickly, by clicking on them with the mouse. Navigate through the curves on the plot layer with 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.

3.8.9 Data -> Select Data Range (Alt-S)

Shows two vertical line cursors that are used for selecting a data range when performing analysis operations. Only one cursor is active at a time. The active cursor is red, the other is black. You change which cursor is active with the Left and Right arrow keys. Move the active cursor with the arrows keys while keeping the Ctrl key pressed or, more easily and quickly, by clicking on a curve point. Note that the mouse cursor shape changes to a rectangular target while inside the active plot canvas. 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.

3.8.10 Data -> Screen Reader

Opens the Data Display toolbar and changes the mouse cursor shape to a small cross target. The coordinates of the cursor with respect to the axes of the active plot layer are displayed in a text field that floats along with the mouse cursor. Clicking the left mouse button at any point will draw a red crosshair at that point and copy the x and y coordinates of that point into the data display.

3.8.11 Data -> Draw Data Points

This command is very similar to the **Screen Reader** command but with the addition of the ability to add data points to a plot by double clicking at each of the desired points. Each invocation of this command will create a new table titled "DrawN", where N is an integer number that is incremented after each new table addition. These tables can be renamed and reformatted after creation using the standard window manipulation commands.

3.8.12 Data -> Move Data points (Ctrl-Alt-M)

Allows modification of the position of data points on the active plot layer by simple drag-and-drop. When selecting this command, you will be asked to confirm that you understand that any changes you make will automatically modify the data in the corresponding tables (as well as all plots depending on that data). The coordinates of the cursor with respect to the axes of the plot layer are displayed in a text field that floats along with the mouse cursor. However, the Data Display toolbar is also opened to provide a higher resolution display the new coordinates.

3.8.13 Data -> Remove Bad Data Points (Alt-B)

Allows removal of data points from the active plot layer by double-clicking on them. When selecting this command, you will be asked to confirm that you understand that any changes you make will also modify the data in the corresponding tables (as well as all plots depending on that data). The coordinates of the cursor with respect to the axes of the plot layer are displayed in a text field that floats along with the mouse cursor. The coordinates of all points selected for removal are shown in the Data Display toolbar.

3.8.14 Data -> Remove Bad Data Points

Allows dragging an entire curve on the selected plot layer. Left Click and hold down the mouse button on any point on a curve to select the curve for dragging. A red cross-hair will be drawn at the point to indicate that the curve is selected. The curve may then be dragged around on the plot layer with the mouse as long as the left mouse button is held down. Releasing the mouse button will drop the curve at the new location. This process moves all the data points and changes the coordinates in the associated table. When selecting this command, you will be asked to confirm that you understand that dragging curves will modify the data in the corresponding tables (as well as all other plots depending on that data). The coordinates of the cursor with respect to the axes of the plot layer are displayed in a text field that floats along with the mouse cursor.

3.9 The Analysis Menu

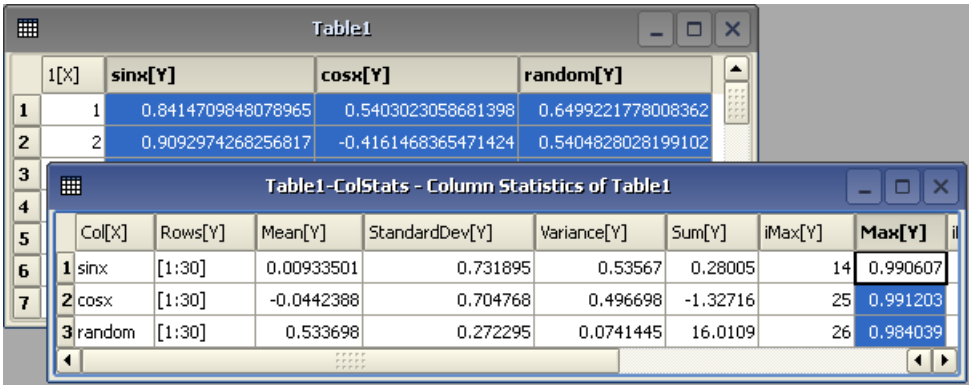
The commands available in this menu change depending upon whether a table or a plot is selected.

3.9.1 Commands for the analysis of data in tables

3.9.1.1 Descriptive Statistics

3.9.1.1.1 Statistics on Columns

The Analysis -> Descriptive Statistics -> **Statistics on Columns** command creates a new table providing basic statistical information about the selected columns in the active table: average, variance, standard deviation, max value, etc...



If you select several columns in one table, one line of statistical information will be created for each selected column. However, you can not select columns from different tables and obtain one single table of statistics.

3.9.1.1.2 Statistics on Rows

The Analysis -> Descriptive Statistics -> **Statistics on Rows** command creates a new table providing basic statistical information about the selected rows in the active table: average, variance, standard deviation, max value, etc. Prior to version 0.9.8.4, statistics were calculated on data in all columns in a selected row which contained numeric data of any kind, whether the column was visible or not. As of version 0.9.8.4, statistics are calculated only on numeric data in visible columns. Usually, statistics should be calculated on data of a similar type - Y values for example. This feature permits selection of only those columns that should be included in the statistical calculation.

See the Statistics on Columns command command for details on exactly what statistical values are calculated.

3.9.1.1.3 Frequency Count

The Analysis -> Descriptive Statistics -> **Frequency Count** command calculates the frequency distribution of the data in the selected column. A dialog is popped up which allows selecting the data range and the bin width. The results are placed in a newly created table. This table will contain 4 columns with rows for each bin. These rows contain 1) the center value of the bin, the frequency (count) for the bin, the lower limit of the bin, and the cumulative frequency up to that bin. If more than one column is selected, an error dialog will pop-up.

3.9.1.1.4 Normality Test

The Analysis -> Descriptive Statistics -> **Normality Test** command computes a Shapiro-Wilk test on the selected column to determine how well the column conforms to a normal distribution. For a detailed discussion of this topic, see the Wikipedia article on Normality Testing.

3.9.1.2 Hypothesis-Testing

3.9.1.2.1 One Sample t-Test

The Analysis -> Hypothesis Testing -> **One Sample t-Test** command performs a Student's t-test on a the selected column of data to estimate the statistical likelihood that the mean has a specific value. It is assumed that the data are normally distributed. The results of the test are placed in the Results Log. The Wikipedia article on the Student's t-test makes for excellent reading on this topic.

3.9.1.2.2 Two Sample t-Test

The Analysis -> Hypothesis Testing -> **Two Sample t-Test** command performs a Student's t-test on two selected columns of data to estimate the statistical likelihood that the means are the same. The results of the test are placed in the Results Log. See the Wikipedia article on the Student's t-test for more information.

3.9.1.3 ANOVA

3.9.1.3.1 One-Way ANOVA

The Analysis -> ANOVA -> **One-Way ANOVA** command performs a One-way analysis of variance on a the selected column of data. The results of the test are placed in the Results Log. The Wikipedia article on the One-way ANOVA for a more detailed discussion.

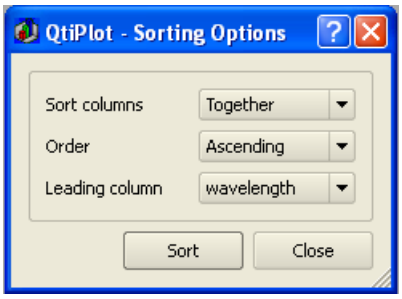
3.9.1.3.2 Two-Way ANOVA

The Analysis -> ANOVA -> **Two-Way ANOVA** command performs a Two-way analysis of variance on a the selected columns of data. The results of the test are placed in the Results Log. See the Wikipedia article on the Analysis of Variance for more details about this topic.

3.9.1.4 Sort Column

The Analysis -> **Sort Column** command sorts the selected columns. If more than one column is selected, you have the option to sort them:

- separately: each column will be sorted independently in ascending or descending order
- together: the column selected as the *leading column* will be sorted into ascending or descending order. The other selected columns will be sorted so as to keep the rows unchanged.



3.9.1.5 Sort Table

The Analysis -> **Sort Table** command functions in the the same manner as the Sort Column command, except that it operates on all columns of the active table.

3.9.1.6 Normalize

These commands are used to normalize data in tables. In this case, normalization is the process of dividing each entry in the column by the column's maximum positive value, making the maximum range value equal to 1. Data is not re-centered so negative values will remain negative and the minimum value may be less than -1. Columns are normalized separately. The command does not create new columns for the normalized data but replaces the values in the selected columns with their normalized values. There are 2 variants of this command:

3.9.1.6.1 Normalize -> Columns

The Analysis -> Normalize -> **Normalize -> Columns** command normalizes only the selected column or columns. If you want the normalized data in a new column, use the **Add Column** command to create a new column, fill it using a "copy/paste" sequence and then normalize the new column.

3.9.1.6.2 Normalize -> Table

The Analysis -> Normalize -> **Normalize -> Table** command normalizes all the columns of the table. It is not a global normalization of all values of the table: each column is normalized separately.

3.9.1.7 Differentiate Column

The Analysis -> **Differentiate Column** command computes the derivative of the selected Y column using centered finite differences. A new plot is created displaying the result of the numerical differentiation.

This command creates a new (hidden) table that contains one column of X-values and one column of derivatives of the Y-values.

3.9.1.8 Integrate Column

The Analysis -> **Integrate Column** command computes the integral of the selected Y column using the trapezoidal. A new plot is created displaying the result of the numerical integration.

This command creates a new (hidden) table that contains one column of X-values and one column of integrated Y-values.

3.9.1.9 FFT...

The Analysis -> **FFT...** command 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.

3.9.1.10 Correlate

The Analysis -> **Correlate** command computes the cross-correlation of the two selected columns. See the correlate section of the Analysis chapter for more details.

3.9.1.11 Autocorrelate

The Analysis -> **Autocorrelate** command computes the cross-correlation of the selected column with itself (auto-correlation). See the correlate section of the Analysis chapter for more details.

3.9.1.12 Convolute

Does a convolution of two selected columns. The first one being the response and the second the signal. See the convolution section of the Analysis chapter for more details.

3.9.1.13 Deconvolute

The Analysis -> **Deconvolute** command computes the deconvolution of two selected columns. The first one being the response and the second the signal. See the deconvolution section of the Analysis chapter for more details.

3.9.1.14 Fit Wizard... (Ctrl-Y)

Opens the Non-linear Fit dialog, allowing you to choose the curve to fit, the algorithm and tolerance to use, and the number of iterations to be performed. You also enter the analytical function, 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.

3.9.2 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-up allowing you to select the curve you want to analyze.

In most cases (except for integration), a new red curve is added to the active plot layer and a new table containing the data used to plot this curve is added to the workspace. Useful information about the operation performed will be shown in the Results Log display.

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

3.9.2.1 Translate

This next group of are 2 commands are used for translating curves horizontally and vertically.

3.9.2.1.1 Vertical

The Analysis -> **Translate -> Vertical** command is used to move a curve in the vertical direction. When the command is selected, the cursor changes to a box and cross with an X-Y coordinate display which floats along with the cursor. You then select 2 points. The first must be a point on a curve. When the first points is selected, a crosshair will be drawn to indicate it's location. The second point can then be selected anywhere in the active graph region. The vertical distance between the selected points will be calculated, and the selected curve will be translated by this amount. The net result is to place the selected graph point at the new vertical location. Horizontal distance is ignored. This allows selection of the second point at one of the vertical axes, facilitating alignment of the selected graph point with some value on that axis. The second point may also be another curve point (either on the same or a different curve), a feature which makes it easy to align two or more curves in the vertical direction. *Warning:* the underlying curve's data is modified by this operation. No other warning will be given.

3.9.2.1.2 Horizontal

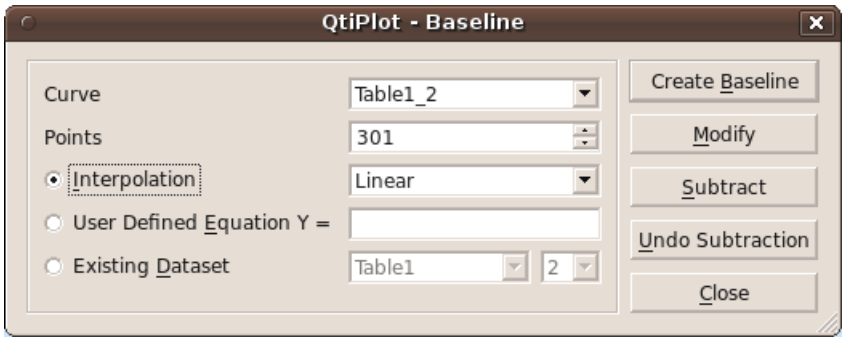
The Analysis -> **Translate -> Horizontal** command is used to move a curve in the horizontal direction. When the command is selected, the cursor changes to a box and cross with an X-Y coordinate display which floats along with the cursor. You then select 2 points. The first must be a point on a curve. When the first points is selected, a crosshair will be drawn to indicate it's location. The second point then can be selected anywhere on the active graph region. The horizontal distance between the selected points will be calculated, and the curve will be translated horizontally by this amount. The net result is to place the selected graph point at the new location. Vertical distance is ignored. This allows selection of the second point at one of the horizontal axes, facilitating alignment of the selected graph point with some value on that axis. The second point may also be another curve point (either on the same or a different curve), a feature which makes it easy to align two or more curves in the horizontal direction. *Warning:* the underlying curve's data is modified by this operation. No other warning will be given.

3.9.2.2 Subtract

The **Subtract** group is also used to translate curves, however, in these cases, some functional relationship is used.

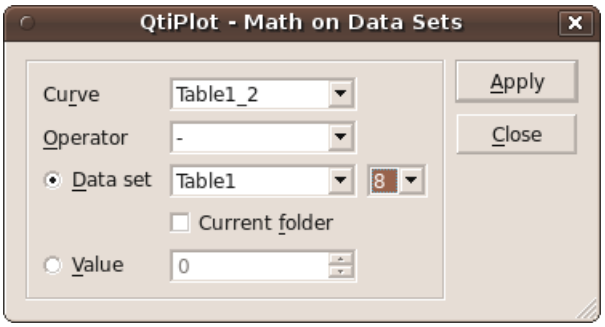
3.9.2.2.1 Subtract -> Baseline

The Analysis **Subtract -> Baseline** command opens the *Baseline* dialog.



3.9.2.2.2 Subtract -> Reference Data

The Analysis **Subtract -> Reference Data** command opens the *Math on Data Sets* dialog.



3.9.2.2.3 Subtract -> Straight Line

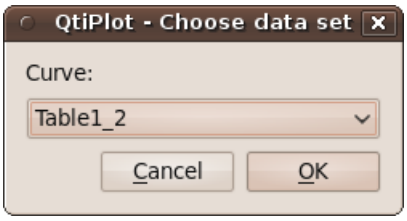
The Analysis **Subtract -> Straight Line** command operates in a manner similar to the Analysis -> Subtract -> Baseline command. However, in this case, the two points which are selected define the end-points of a straight line which is subtracted from *all* the curves on the layer.The cursor changes to a cross target with an X-Y coordinate display which floats along with the cursor, similar to that used in the Analysis -> Subtract -> Baseline command. When the first point is selected, it will be marked with a small circle and crosshairs. As soon as the second point is selected, the line between the two points will be subtracted from the curves. *Warning:* the underlying curves' data is modified by this operation. No other warning will be given.

3.9.2.3 Differentiate

the Analysis -> **Differentiate** command creates a new plot displaying the resulting curve of the numerical differentiation. The computation of the derivative is done by centered finite differences. A new table is created which contains one column for X-values and one column for derivatives of Y-values. It also creates a new plot of the derivative.

3.9.2.4 Integrate Curve

the Analysis -> **Integrate Curve** command performs a piecewise numerical integration of a curve on the selected layer. Curves are integrated using the trapezoidal rule. There must be at least one curve on the layer. If there are multiple curves, a dialog will be opened asking you to select which curve to use for the integration:



After selecting a curve, you may either proceed by clicking "OK" or abort with "Cancel". If there is only one curve on the layer, the integration is performed immediately upon selecting the **Integrate Curve** command. A new Graph Window will be created containing a plot of the integrated curve, the data for which is stored in a new (hidden) table.

3.9.2.5 Integrate Function...

The Analysis -> **Integrate Function...** command provides the capability of performing the numerical integration of a user defined function. The Function Integration dialog is opened when this command is executed. This dialog provides for the definition of the function, the variable of integration, and the parameters that control the integration. Integration is performed using the QAGS algorithm from the GNU Scientific Library. For many functions, all you need do is set the limits of integration and click the "Integrate" button. The default values for the other parameters will usually produce excellent results. For some functions, especially those containing singularities and discontinuities, you may need to experiment with the number of sub-intervals and the tolerance limit to obtain acceptable results. However, a detailed discussion of the use of these 2 parameters is beyond the scope of this manual. If the *Plot Area* option is checked, a curve of the integrated function will be added to the currently active layer. In any case, an entry is made in the Log Panel which contains the numerical result of the computed definite integral over the specified interval.

3.9.2.6 Smooth

There are 4 variants of the **Smooth** menu item:

3.9.2.6.1 Savitski-Golay

The Analysis -> **Smooth -> Savitsky-Golay...** command performs a smoothing of the selected curve using 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 polynomial. They depend on the number of points used for the smoothing of the curve and the order of the polynomial. 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 specification of the curve which will be smoothed, the order of the polynomial, 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.

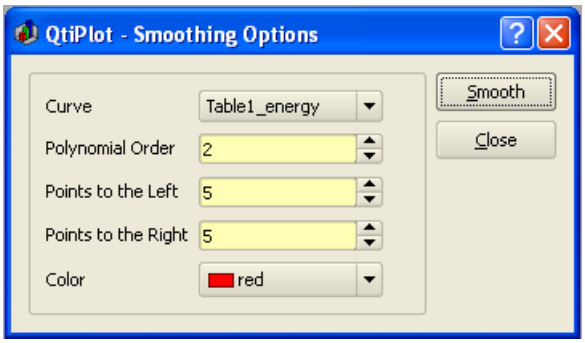


Figure 3.1: The **Smooth -> Savitsky-Golay...** dialog.

3.9.2.6.2 Moving Window Average...

The Analysis -> **Smooth -> Moving Window Average...** 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 $z_i=f(x_i)$ will be. The dialog allows specification of 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.

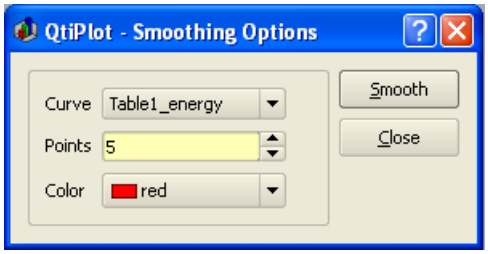


Figure 3.2: The **Smooth -> Moving Window Average...** dialog.

3.9.2.6.3 Lowess...

The Analysis -> **Smooth -> Lowess...** 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.

The 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%). Values of *f* closer to 1 yield smoother curves. The *iterations* parameter specifies the number of times to run the algorithm runs over the entire data set, each time refining the local weights. In most cases, two iterations will be sufficient.

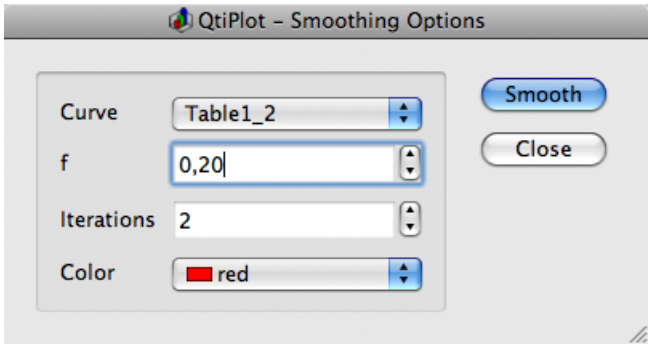


Figure 3.3: The **Smooth -> Lowess...** dialog.

3.9.2.7 FFT Filter

The Analysis -> **FFT Filter** menu item provides Lowpass, Highpass, Bandpass and Bandreject filter topologies:

3.9.2.7.1 Low Pass;

The Analysis -> **FFT Filter -> Low Pass...** command uses an FFT based digital filter to attenuate the high frequencies present in an input signal. See the filtering section for more details. A dialog box will be opened in which you can select the curve (input signal) to filter and the cut-off frequency of the filter.

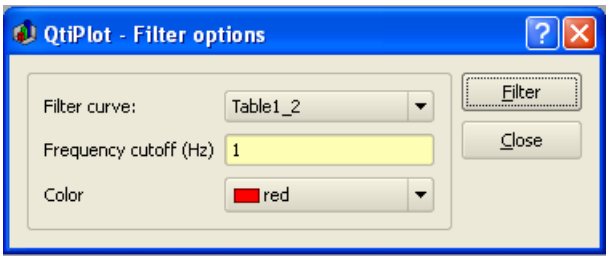


Figure 3.4: The **FFT Filter -> Low Pass...** dialog.

This command creates a new table containing the filtered data, and adds a new curve to the current layer. The curve is a plot of the filtered data.

3.9.2.7.2 High Pass...

The Analysis -> **FFT Filter -> High Pass...** command uses an FFT based digital filter to attenuate the low frequencies present in an input 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.

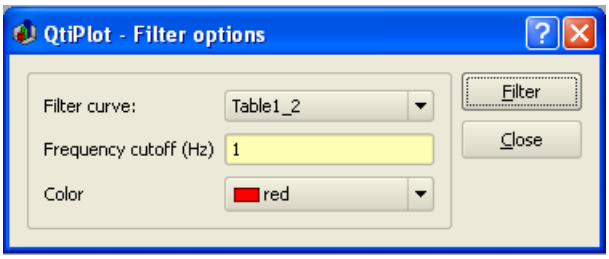


Figure 3.5: The **FFT Filter -> High Pass...** dialog.

This command creates a new table containing the filtered data, and adds a new curve to the current layer. The curve is a plot of the filtered data.

3.9.2.7.3 Band Pass...

The Analysis -> **FFT Filter -> Band Pass...** command uses an FFT based digital filter to attenuate both high and low frequencies present in an input signal. See the filtering section for more details. A dialog box will be opened in which you can select the curve to filter and both the low and high cutoff frequencies of the filter.

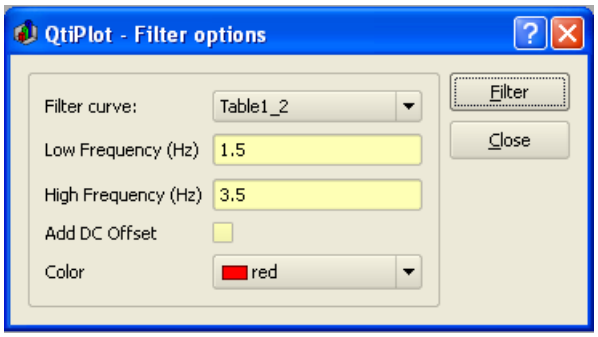


Figure 3.6: The **FFT Filter -> Band Pass...** dialog.

This command creates a new table containing the filtered data, and adds a new curve to the current layer. The curve is a plot of the filtered data.

3.9.2.7.4 Band Block...

The Analysis -> **FFT Filter -> Band Pass...** command uses an FFT based digital filter to remove a band of frequencies from a signal while leaving those frequencies above and below the stop band. See the filtering section for more details. A dialog box will be opened in which you can select the curve to filter and both the lower and upper stop-band frequencies of the filter.

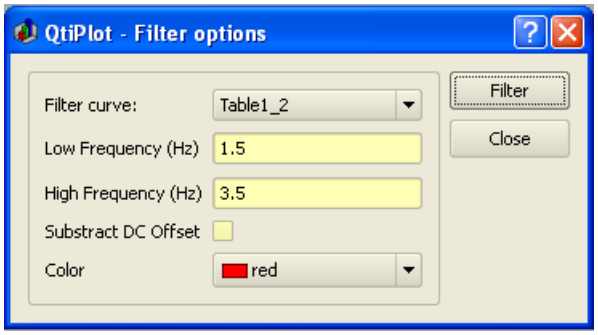


Figure 3.7: The **FFT Filter -> Band Block...** dialog.

This command creates a new table containing the filtered data, and adds a new curve to the current layer. The curve is a plot of the filtered data.

3.9.2.8 Analysis -> Interpolate...

The Analysis -> **Interpolate...** command performs an interpolation using the selected method. The curve must have enough data points to compute interpolated points, if not a warning message will be popped up.

The available interpolation methods are *Linear* (the curve must contain at least 3 points), *Cubic Spline* (the curve you analyze must contain at least 4 points), *Non-rounded Akime spline* (the curve you analyze must contain at least 5 points). See the Analysis chapter for a comparison of the different methods.

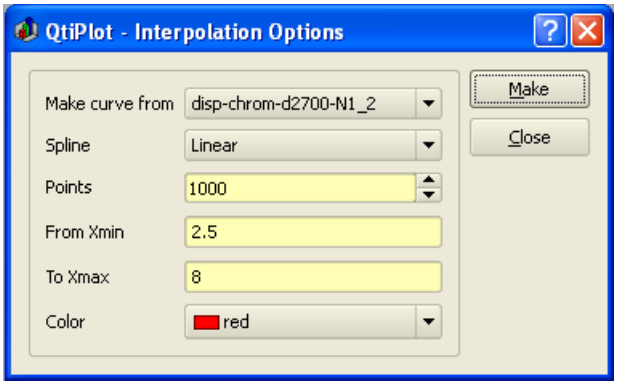


Figure 3.8: The **Interpolate...** dialog.

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

3.9.2.9 FFT...

The Analysis -> **FFT...** command performs a forward or inverse FFT 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.

3.9.2.10 Fit Linear

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

3.9.2.11 Fit Polynomial...

The Analysis -> **Fit Polynomial...** command 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 abscissa limits for the fit.

3.9.2.12 Fit Exponential Decay

Analysis -> **Fit Exponential Decay** provides 3 forms of exponential decay:

3.9.2.12.1 First Order...

The 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.

3.9.2.12.2 Second Order...

The Analysis -> **Fit Exponential Decay -> Second Order...** opens a dialog, allowing you to choose the curve to fit and the initial guesses for the fit parameters.

3.9.2.12.3 Third Order...

The Analysis -> **Fit Exponential Decay -> Third Order...** opens a dialog, allowing you to choose the curve to fit and the initial guesses for the fit parameters.

3.9.2.13 Fit Exponential Growth...

Analysis -> **Fit Exponential Growth...** performs an exponential growth fit to the selected curve.

3.9.2.14 Fit Boltzmann (sigmoidal)

The Analysis -> **Fit Boltzmann (sigmoidal)** performs a fit to a Boltzmann function on the selected curve. It can be used to obtain the correlation equation of an S shaped data set.

3.9.2.15 Fit Gaussian

The Analysis -> **Fit Gaussian** performs a Gaussian fit to the selected curve. It can be used to obtain the correlation equation of a bell shaped data set.

3.9.2.16 Fit Lorentzian

Analysis -> **Fit Lorentzian** performs a Lorentzian fit to the selected curve. It can be used to obtain the correlation equation of a bell shaped data set.

3.9.2.17 Fit Multi-peak ->Gaussian...

The Analysis -> **Fit Multi-peak ->Gaussian...** performs a fit to a sum of N Gaussian functions on the selected curve.

3.9.2.18 Analysis -> Fit Multi-peak -> Lorentzian...

The Analysis -> **Fit Multi-peak -> Lorentzian...** performs a fit to a sum of N Lorentzian functions on the selected curve.

3.10 The Table Menu

This menu is only active when a table is selected.

3.10.1 Set Column As

Selecting the **Set Column As** item opens a sub-menu of command that are used to define the kind of data stored in the various columns of a table.

3.10.1.1 Set Column As -> X

Define the selected column as abscissa for the plots. You can define more than one column as X-values in a table, they are referenced as X1, X2, etc.

3.10.1.2 Set Column As -> Y

For 2D plots, this command defines the selected column as Y-values for the plots. For 3D plots, Y columns can be used as the second abscissa.

3.10.1.3 Set Column As -> Z

For 3D plots, Z columns will be used as plotted values.

3.10.1.4 Set Column As -> X error

Define the selected column for use as the error bar width for the abscissae.

3.10.1.5 Set Column As -> Y error

Define the selected column for use as the error bar heights for the Y-values.

3.10.1.6 Set Column As -> Read-only

Set the selected columns as read-only.

3.10.1.7 Set Column As -> Read/Write

Restore write access to the selected columns.

3.10.1.8 Set Column As -> label

Sets the selected columns as *label* columns. Can be used for comments and row descriptions.

3.10.1.9 Set Column As -> Disregard

No special function is assigned. The selected column can be used in different ways in several plots (as X values, Y values, etc). These columns are disregarded in statistical calculations.

3.10.2 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.

3.10.3 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.

3.10.4 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.

3.10.5 Fill column with

These commands are used to fill selected columns with special values:

3.10.5.1 Fill Column With -> Row Numbers

Each element in the column is filled with its corresponding row number.

3.10.5.2 Fill Column With -> Random Numbers

Each element in the column is filled with a random value between 0 and 1.

3.10.5.3 Fill Column With -> Normal Random Numbers

The rows in the selected column are filled with normally distributed random values calculate using the Ziggurat method with a mean of 0.0 and a standard deviation of 1.0. The computational routine is from the Gnu Scientific Library (look here for more details).

3.10.6 Clear

Removes all the values of the selected column.

3.10.7 Add Column

Adds a new column to the table. Regardless the selected column, new columns are inserted to the right of the rightmost column in the table.

3.10.8 Set Columns...

Used to define the number of columns in the table. Columns are added/removed from the right hand side of the table. Be Careful! If you decrease the number of columns in a table, any data contained in the removed columns will be lost!

3.10.9 Hide Selected Columns

The **Hide Selected Columns** command hides all the selected columns. The remaining visible columns are grouped together into a single block. Hidden columns may be shown using the **Show All Columns** command.

3.10.10 Show All Columns

The **Show All Columns** command unhides any hidden columns in the selected table.

3.10.11 Set Optimal Column Width

The **Set Optimal Column Width** command resets the width of all selected columns to a value that is optimal for the data that is contained in the column. Optimal width is considered to be just wide enough to show all digits.

3.10.12 Move to First

Moves the selected column to the beginning of the table.

3.10.13 Move Left

Moves the selected column to the left.

3.10.14 Move Right

Moves the selected column to the right.

3.10.15 Move to Last

Moves the selected column to the end of the table.

3.10.16 Swap columns

Swaps the selected columns.

3.10.17 Set Rows...

Allows direct definition of the number of rows in the table. Rows are added/removed from the end of the table. Be Careful! If you decrease the number of rows in a table, any data contained in removed rows will be lost.

3.10.18 Delete Rows Interval...

A dialog is opened that permits selection, and subsequent deletion, of a range of rows selected by row index number.

3.10.19 Move Row >

These commands are used to move selected rows up or down in a table:

3.10.19.1 Move Row -> UP

The selected row is moved up one place in the table.

3.10.19.2 Move Row -> DOWN

The selected row is moved down one place in the table.

3.10.20 Go to Row... (Ctrl-Alt-G)

This command opens a dialog which allows you to select the row index that will become the current row in the selected table or matrix.

3.10.21 Go to Column... (Ctrl-Alt-C)

This command opens a dialog which allows you to select the column index that will become the current column in the selected table or matrix.

3.10.22 Extract Data...

The **Extract Data...** command opens a dialog which allows you to define a set of conditions that are used to filter the data in the currently active table. When a condition has been defined, *Applying* the condition will create a new table into which all rows that meet the condition will be copied. The original table is unchanged. For example, the condition *col("1")>=.1* will generate a new table that contains all the rows from the active table which have a value greater than 0.1 in column 1.

3.10.23 Convert to Matrix

This command is used to convert a table into a matrix. It is mainly used to import data from files into a matrix: first import the data into a table, and then use this command to convert the table into a matrix.

3.11 The Matrix Menu

This menu is only active when a matrix is selected.

3.11.1 Set Properties...

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

3.11.2 Set Dimensions... (Ctrl-D)

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.

3.11.3 Set Values... (Ctrl-Q)

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.

3.11.4 Recalculate (Ctrl-Return)

This command allows you to recalculate the matrix cell values using a predefined formula. Useful when the formula is changed.

3.11.5 Rotate 90 (Ctrl-Shift-R)

This command performs a clockwise 90 degrees rotation of the active matrix.

3.11.6 Rotate -90 (Ctrl-Alt-R)

This command performs a counterclockwise 90 degrees rotation of the active matrix.

3.11.7 Flip V (Ctrl-Shift-V)

Flips the selected matrix vertically.

3.11.8 Flip H (Ctrl-Shift-H)

Flips the selected matrix horizontally.

3.11.9 Expand...

Opens the matrix resampling dialog. Expanding operation is preselected in the dialog.

3.11.10 Shrink...

Opens the matrix resampling dialog. Shrinking operation is preselected in the dialog.

3.11.11 Smooth

Matrix smoothing is performed by shrinking and then expanding the matrix using bilinear interpolation. If the number of columns or rows is less than 32, the matrix is first expanded so that the row number and the column number are both twice of the original. Then the expanded matrix is shrunk to the original size. Through this process of expanding and shrinking, the size of the output matrix will be exactly the same as the original matrix. However, the data will be much smoother. If both the number of columns and the number of rows in the original matrix are greater than 31, the matrix is first shrunk and then expanded to obtain the smoothed matrix.

3.11.12 Transpose

Perform a Transpose operation on the selected matrix.

3.11.13 Invert

Invert the selected matrix.

3.11.14 Determinant

Compute the determinant of the selected matrix.

3.11.15 Go To Commands

The Go to Row... and Go to Column... commands are the same as those described in the Table menu

3.11.16 View Commands

This group of commands allows you to choose how the matrix will be displayed.

3.11.16.1 Image mode (Ctrl-Shift-I)

Displays the selected matrix as an image.

3.11.16.2 Data mode (Ctrl-Shift-D)

Displays the selected matrix as a data table.

3.11.17 Palette

3.11.17.1 Gray Scale Map

If the selected matrix is viewed as an image, this command sets the palette to a gray scale.

3.11.17.2 Rainbow

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

3.11.17.3 Custom

Opens a dialog allowing customization the palette used by the selected matrix when the matrix is displayed as an image.

3.11.18 Show Column/Row (Ctrl-Shift-C)

When this option is checked, the horizontal and vertical headers of the selected matrix will display the column/row indexes.

3.11.19 Show X/Y (Ctrl-Shift-X)

When this option is checked, the horizontal and vertical headers of the selected matrix will display the x/y coordinates.

3.11.20 Convert to Spreadsheet

Convert the selected matrix into a table.

3.12 The Format Menu

This menu is only active when a plot is selected.

3.12.1 Plot...

For classical 2D plots, opens the format plot dialog with the general plot options tab selected. It is used to customize line styles and colors of the plot frame, etc.

For surface plots, 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.

3.12.2 Curves...

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

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

3.12.3 Scales...

Opens the format plot dialog with the scales tab selected. Used to customize the ranges on the different axes. Note that any modification in an associated table, or of the plotted curves, will result in a reset of these scales to the default values.

For surface plots, this command opens the surface plot options with the scales options tab selected.

3.12.4 Axes...

Opens the format plot dialog with the axes tab selected. Used to customize the settings for the different axes, such as axis/tick size and color, axis labels, etc.

For surface plots, this command opens the surface plot options with the axis options tab selected.

3.12.5 Grid...

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

This menu item is not shown for surface plots.

3.12.6 Title...

Opens a text options dialog, which permits modification of text and properties (color, font, alignment) of the plot title.

For surface plots, this command opens the surface plot options with the title options tab selected.

3.13 The Windows Menu

In addition to the items listed below, this menu will also display a list containing 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 on the remaining windows.

3.13.1 Folders

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

3.13.2 Cascade

Arranges the visible windows in the project in a cascading style.

3.13.3 Tile

Tiles the visible windows in the project.

3.13.4 Next (F5)

Makes the next visible window in the workspace stack the active window.

3.13.5 Previous (F6)

Makes the previous visible window in the workspace stack the active window.

3.13.6 Rename Window

Opens a dialog permitting you to change the title of the currently active window.

3.13.7 Duplicate

Clones the active window.

3.13.8 Script Window (F3)

Opens the script console window.

3.13.9 Window Geometry...

Opens a dialog used to change the size and the position of the active window. The size of any contained plots will be scaled to the new window size.

3.13.10 Hide Window

Hides the active window. A hidden window can be made visible again using the Project explorer.

3.13.11 Close Window (Ctrl-W)

Closes the active window. A dialog will pop-up asking you to confirm the operation if the corresponding option in the Preferences dialog ("Confirmations" tab) is checked.

3.13.12 Numbered Window List

A numbered list of the current folder's windows appears at the bottom of the Window Menu. This list may contain up to 9 entries. Clicking on one of the entries in this list will activate that window (that is, display the window, and unhide it if necessary. If there are more than 9 entries, the *More Windows...* command will be added to the Window Menu immediately following the last item in the numbered window list. Selecting the *More Windows...* command opens the Project Explorer, which provides access to all windows in all folders in the project.

3.14 Customization of 3D plots

These commands are not available in any menu or by using any keyboard shortcut. However, they can be accessed through the 3D toolbar.

3.14.1 Frame

This command can be accessed by a click on the 

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

3.14.2 Box

This command can be accessed by a click on the 

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

3.14.3 No axes

This command can be accessed by a click on the 

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

3.14.4 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 on the plane defined by $y=y_{min}$.

3.14.5 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 on the plane defined by $y=y_{max}$.

3.14.6 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 on the plane defined by $x=x_{min}$.

3.14.7 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 on the plane defined by $x=x_{max}$.

3.14.8 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 on the plane defined by $z=z_{max}$.

3.14.9 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 on the plane defined by $z=z_{min}$.

3.14.10 Enable perspective

This command can be accessed by a click on the 

Enables/Disables 3D perspective mode.

3.14.11 Reset rotation

This command can be accessed by a click on the 

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

3.14.12 Fit frame to window

This command can be accessed by a click on the 

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

3.14.13 Bars Style

This command can be accessed by a click on the 

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

3.14.14 Dots

This command can be accessed by a click on the 

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

3.14.15 Cones

This command can be accessed by a click on the 

If the active 3D plot is a 3D scatter, this command permits modification of the style used to draw data points as cones. It it then possible to modify the drawing parameters of the cones by double clicking on the plotting area.

3.14.16 Cross Hairs

This command can be accessed by a click on the 

If the active 3D plot is a 3D scatter, this command permits modification of the style used to draw data points as cross-hairs. It it then possible to modify the drawing parameters of the crosses by double clicking on the plotting area.

3.14.17 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 be drawn as a simple wireframe.

3.14.18 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 be drawn as a wireframe. A computation of hidden lines is performed.

3.14.19 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 be drawn as polygons.

3.14.20 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 be drawn as polygons with a mesh.

3.14.21 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.

3.14.22 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.

3.14.23 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.

3.14.24 Animation

This command can be accessed by a click on the

Enables/disables animation.

Chapter 4

The Toolbars

Groups of related commands are collected together in a number of toolbars. All toolbars can be moved and docked in a more convenient location (left, right or bottom sides of the application window) or moved onto the desktop (outside the main window) using drag-and-drop, using their left side handle. Toolbars are automatically enabled/disabled depending on the currently active window. For example, if the current window is a table, the Table toolbar will be enabled and all the other toolbars will be automatically disabled.

The same approach is used for showing/hiding toolbars: if there are no visible tables in the workspace, the Table toolbar will be automatically hidden. It will be shown again when the user adds a new table into the project. Toolbars can also be manually shown/hidden at any time by right-clicking on the **Toolbars...** command from the View menu or entering Ctrl-Shift-T and then checking/unchecking the corresponding item in the pop-up menu.

Visible toolbars will be shown in one of two possible formats depending upon the space available at the docking location. By default, when there is sufficient room, all tool icons will be shown in a single row or column. If there is not sufficient room to show all icons in one row, the toolbar will be collapsed to the extent necessary to fit the toolbar into the available space. In this case, the ">>" symbol will be shown to indicate that there are hidden command icons. As long as the toolbar is not disabled, left clicking on the ">>" symbol will temporarily expand the toolbar and show the hidden commands, which may then be left clicked to execute them. Moving the mouse pointer off of the toolbar will automatically collapse it. Note that toolbars located on the desktop are always shown expanded.

4.1 The Edit Toolbar

Figure 4.1: The QtiPlot Edit Toolbar

4.2 The File Toolbar

The *File Toolbar* provides single-click access to commands from the File menu. Refer to this section for a more complete description of these commands.

Figure 4.2: The QtiPlot File Toolbar

Icon	Command	Key	Description
	Undo command	Ctrl-Z	Undo the last command, this feature doesn't work for plot modifications. See the Undo command command for more details.
	Redo command	Ctrl-Shift-Z	Redo the last command, this feature doesn't work for plot modifications. See the Redo command command for more details.
	Cut Selection	Ctrl-X	Cut the current selection.
	Copy Selection command	Ctrl-C	Copy the current selection.
	Paste Selection command	Ctrl-V	Paste the current selection.
	Delete Selection command	Del	Delete the current selection.

Table 4.1: Edit toolbar commands.

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	Ctrl-M	Create a new matrix.
	New -> New Note command		Create a new note window.
	New -> New Graph command	Ctrl-G	Creates a new, empty graph window (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.
	Open Template command		Opens an existing QtiPlot template file.
	Open Excel command	Ctrl-Shift-E	Imports an Excel workbook into new tables (and plots if Excel is installed).
	Open ODF Spreadsheet command	Ctrl-Alt-S	Imports an ODF spreadsheet into new tables.
	Append Project... command	Ctrl-Alt-A	Appends an existing QtiPlot project file as a new folder to the current project.
	Save Project command	Ctrl-S	Saves the current project.
	Save as Template command		Saves the current graph window as a template.
	Import -> Import ASCII... command	Ctrl-K	Imports ASCII files.
	Duplicate command		Clones the active window.
	Print command	Ctrl-P	Prints the active window.
	Print Preview command		Previews the active window for printing.
	Export to PDF command	Ctrl-Alt-P	Exports the active window formatted as a PDF file.
	Project Explorer	Ctrl-E	Shows or hides the project explorer.
	Results log command		Shows or hides the results window.
	Script Window command	F3	Shows the script window.

Table 4.2: File toolbar commands.

4.3 The Plot Toolbar

This toolbar is only active when a graph window is selected. It provides single-click access to the commands of the Graph menu and of the Data menu which are used for the modification of plots and the data points of the plots.

Figure 4.3: The QtiPlot Plot Toolbar

Icon	Command	Key	Description
	Add Layer	Alt-L	Adds a new layer to the active plot window.
	Add Empty Inset Layer		Adds an empty inset layer to the active graph window.
	Add Inset Layer With Curves		Adds an inset layer to the active graph window that contains copies of any curves in the active plot layer.
	Arrange Layers	Shift-A	Opens a dialog for rearranging all layers of the active graph window.
	Automatic Layout		Automatically rearranges all layers of the active graph window.
	Extract to Layers		Extracts datasets (curves) from the active layer onto separate layers.
	Extract to Graphs		Extracts all layers in the active graph window to separate graphs.
	Add Error Bars...	Ctrl-B	Adds error bars to a curve on the active plot layer.
	Add/Remove Curves...	Alt-C	Adds curves to, or removes curves from, the active plot window.
	Add Function...	Ctrl-Alt-F	Adds a curve based on a function to the active plot layer.
	New Legend	Ctrl-L	Adds a new legend to the active plot layer.
	Rescale To Show All	Ctrl-Shift-R	Rescales the axes to show all data points.
	Disable tools		Reverts to normal pointer mode. Useful when you have selected another pointer mode on a plot layer, such as the data reader.
	Zoom Commands		Selects the last used zoom tool or selects a new one from a drop down list.
	Data Reader	Ctrl-D	Switches the data display into Data Reader mode.
	Select Data Range	Alt-S	Switches the active plot layer into **Select Data Range** mode.
	Screen Reader		Switches the active plot layer into **Screen Reader** mode.
	Draw Data Points		Adds new data points on the active plot layer.
	Move Data points	Ctrl-Alt-M	Allows drag-mode moving of data points on the active plot layer.
	Remove Bad Data Points	Alt-B	Allows removal of data points from the active plot layer.
	Remove Bad Data Points		Drags a selected curve around on plot layer.
Σ	Add Equation	Alt-Q	Adds a new *Tex* formatted equation on the active plot layer.
T	Add Text	Alt-T	Adds a new text element on the active plot layer.
↗	Draw Arrow	Ctrl-Alt-A	Adds a new arrow on the active plot layer.
	Draw Line	Ctrl-Alt-L	Adds a new line on the active plot layer.
	Add Rectangle	Ctrl-Alt-R	Adds a new rectangle on the active plot layer.
	Add Ellipse	Ctrl-Alt-E	Adds a new ellipse (circle) on the active plot layer.
	Add Time Stamp	Ctrl-Alt-T	Adds a time/date label on the active plot layer.
	Add Image	Alt-I	Inserts a new image into the active plot layer.
	Move to Top		Moves the selected item to the top of the layer's z-order.
	Move to Bottom		Moves the selected item to the bottom of the layer's z-order.

Table 4.3: Plot toolbar commands

Icon	Command	Key	Description
	Zoom In/Out and Drag Canvas		Switches the active plot layer into dynamic zoom/drag mode.
	Zoom/Drag Canvas Horizontally		Switches the active plot layer into dynamic horizontal zoom/-drag mode.
	Zoom/Drag Canvas Vertically		Switches the active plot layer into dynamic vertical zoom/drag mode.
	Zoom in	Ctrl-+	Switches the active plot layer into zoom mode.
	Zoom out	Ctrl--	Zooms the active plot layer out one step in the zoom history.

Table 4.4: Plot Toolbar Zoom Commands

4.4 The Table Toolbar

This toolbar provides single-click access to the Plot Menu commands. These are used to plot data from tables.

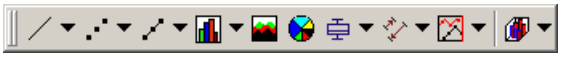


Figure 4.4: The QtiPlot Table Toolbar

Icon	Command	Key	Description
	Line Plots		Line, Horizontal Steps, Vertical Steps
	Scatter Plots		Scatter, Vertical Drop Lines
	Line & Symbol Plots		Line+Symbol, Spline
	Bar Chart Plots		Columns, Rows, Stack Column, Stack Bar
	plot -> Area		Plots selected data using area style.
	plot -> Pie		Plots selected data using pie style.
	Statistical Plots		Box, Histogram, Stacked Histogram, Stem-Leaf
	Vector Plots		Vectors XYXY. Vectors XYAM
	Special Line/Symbol Plots		Double-Y, Waterfall, Zoom, Vert. 2 Layer, Horiz. 2-Layer, 4 Layer, Stacked Layer, Custom
	3D Plots		Bars, Ribbons, Scatter, Trajectory

Table 4.5: Table toolbar commands.

Icon	Command	Key	Description
	plot -> Line		Plots selected data using line style.
	plot -> Horizontal Steps		Plots selected data using horizontal steps style.
	plot -> Vertical Steps		Plots selected data using vertical steps style.

Table 4.6: Line Plots

4.5 The Column Toolbar

This toolbar provides single-click access to the commands of the Table Menu which are used to rearrange columns in the table, fill columns with values and define the plot role of selected columns. A few items from the Analysis menu are also included on this toolbar.

Icon	Command	Key	Description
	plot -> Scatter		Plots selected data using scatter style.
	plot -> Vertical Drop Lines		Plots selected data using vertical drop lines style.

Table 4.7: Scatter Plots

Icon	Command	Key	Description
	plot -> Line+Symbol		Plots selected data using line+symbol style.
	plot -> Spline		Plots the selected data columns using spline style.

Table 4.8: Line & Symbol Plots

Icon	Command	Key	Description
	plot -> Columns		Plots selected data using columns style.
	plot -> Rows		Plots selected data using rows style.
	plot -> Special Bar/Column -> Stack Column		Plots selected data using stack column style.
	plot -> Special Bar/Column -> Stack Bar		Plots selected data using stack row style.

Table 4.9: Bar Chart Plots

Icon	Command	Key	Description
	plot -> Box Plot		Plots selected data using Box style.
	plot -> Histogram		Plots selected data using histogram style.
	plot -> Stacked Histogram		Plots selected data using stacked histogram style.
	plot -> Stem and Leaf		Plots selected data using Stem and Leaf style.

Table 4.10: Statistical Plots

Icon	Command	Key	Description
	plot -> Vectors XYXY		Plots selected data using xyxy vector style.
	plot -> Vectors XYAM		Plots selected data using xyam vector style.

Table 4.11: Vector Plots

Icon	Command	Key	Description
	plot -> Double-Y		Plots selected data using two different y axes.
	plot -> Waterfall		Creates a Waterfall plot from a series of selected columns.
	plot -> Zoom		Plots selected data in one plot layer along with a magnified region of that data in another plot layer on the same graph window.
	plot -> Vertical 2 Layers		Plot 2 vertically arranged layers on the same graph window.
	plot -> Horizontal 2 Layers		Plot 2 horizontally arranged layers on the same graph window.
	plot -> 4 Layers		Plot 4 layers on the same graph window.
	plot -> Stacked Layers		Plot a stacked layer for each selected column, all on the same graph window.
	plot -> Custom Layout...		Opens the Arrange Layers dialog to set up custom layouts.

Table 4.12: Special Line/Symbol Plots

Icon	Command	Key	Description
	plot -> Bars		Plots selected data using 3D bars style.
	plot -> Ribbons		Plots selected data using 3D ribbons style.
	plot -> Scatter		Plots selected data using 3D scatter style.
	plot -> Trajectory		Plots selected data using trajectory style.

Table 4.13: 3D Plots

Figure 4.5: The QtiPlot Column Toolbar

Icon	Command	Key	Description
	Table -> Set Column Values...	Alt-Q	Used for functional assignment of values in selected column.
	Table -> Fill Column With -> Row Numbers		Fills each row in the selected column with it's corresponding row number.
	Table -> Fill Column With -> Random Numbers		Fills each row in the selected column with a random value.
	Table -> Fill Column With -> Normal Random Numbers		Fills each row in the selected column with a normally distributed random value.
	Analysis -> Sort Table		Sorts all columns in the selected table into ascending or descending order.
	Analysis -> Statistics on Columns		Computes statistical parameters on selected columns.
	Analysis -> Statistics on Rows		Computes statistical parameters on the selected rows.
	Table -> Set Column As -> X		Define the selected column as abscissae for plotting.
	Table -> Set Column As -> Y		Define the selected column as an ordinate for plotting.
	Table -> Set Column As -> Z		For 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.
	Table -> Set Column As -> label		Define the selected column as containing labels.
	Table -> Set Column As -> Disregard		This command removes any plot role from 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 -> Set Optimal Column Width		Sets an optimal width on the selected columns.
	Table -> Move Row -> UP		Moves selected row UP one slot in table.
	Table -> Move Row -> DOWN		Moves selected row DOWN one slot in table.
	Table -> Add Column	Alt-C	Adds a new column to the selected table.

Table 4.14: 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 -> 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 4.15: 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 script launchers by defining: 1) a textual description that will be displayed in the destination menu, (2) an icon, (3) a shortcut key sequence and (4) a tool tip. The *Tool Tip Text* is a short description that is displayed near the launcher button when you hover the mouse over it. You must assign unique shortcut key sequences to your custom launchers.

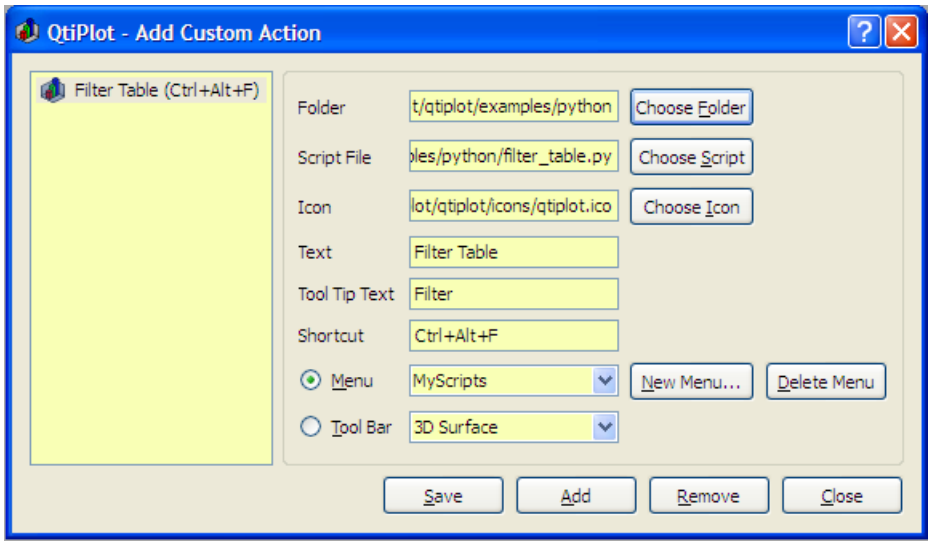


Figure 5.1: The **Add Custom Script Action...** dialog box.

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. Make sure that you have backed up the file if you will ever need it in the future!

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 at each data point.

Note that clicking on the "Close" button will not add any error bars to the plot. You must set up the parameters for the error bars first and then click on the "Add" button. If you click on "Close" without clicking "Add", there will be no error bars drawn.

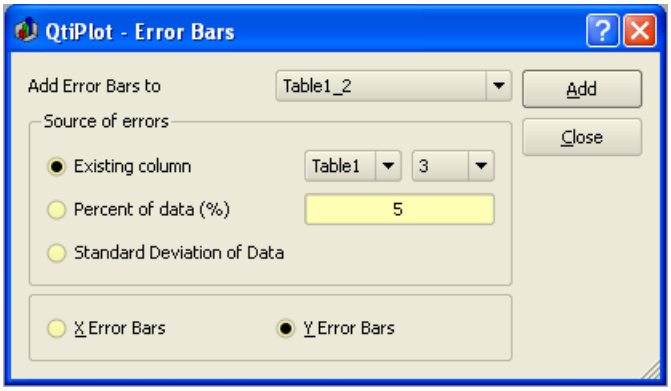


Figure 5.2: The **Add Error Bars...** dialog.

There are three ways to specify the sizes of the bars:

1) From a column of the table: In this case, the values in the selected column are used to compute the error bars. Specifically, if V is the value of a data point, and E the value from the same row of the errorbar column, the size of the corresponding bar will be V-E to V+E.

2) A percentage of the values: This command adds a new column to the table associated with the plot and fills it with result of calculating V*E/100 for each row in the table, where V is the plotted value and E is the selected percentage. This column is then used to plot the error bars in exactly the same manner as in the previous case. The new column can subsequently be edited like any other column. Any changes will be reflected in the size of the corresponding error bars.

3) The standard deviation of the values: This command first calculates the standard deviation of the plotted values. This has meaning only if the data are centered around an average value. As with the previous option, a new column will be created in the active table and each row will be filled with the computed standard deviation. The program does not know and cannot compute individual standard deviations at each point. If this is required, you will need to independently compute these values from multiple samples taken at each x or y value.

If you need to display asymmetric error bars you must define two sets of error bars for the same curve: one for the upper limits (with disabled minus side) and one for the lower limits (with disabled plus side).

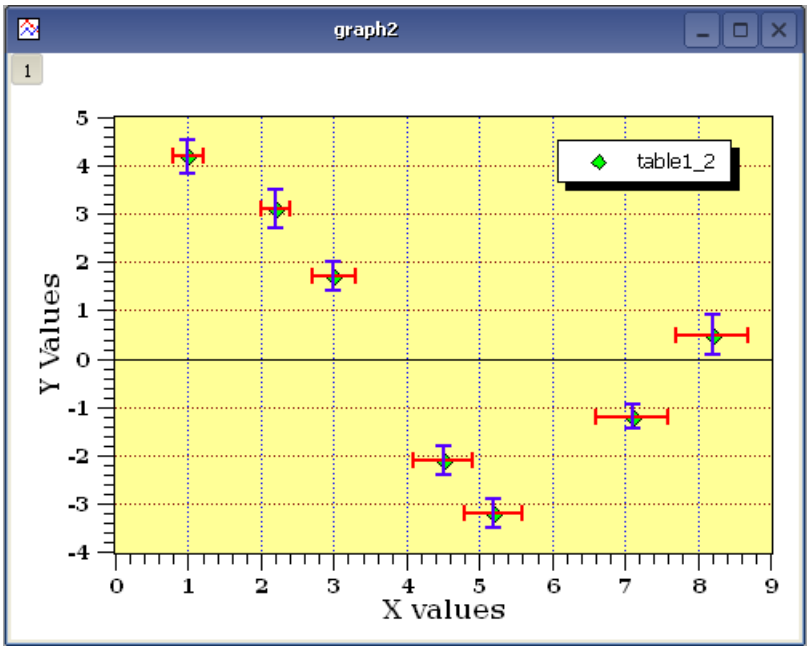


Figure 5.3: Example of a plot with both X and Y Error Bars.

5.3 Add Function

This dialog box is used to add a function curve to the active plot. The plotted function is built with the common operators *, +, /, -, and ^ (for multiplication, addition, division, subtraction and exponentiation, respectively). A complete set of intrinsic functions are available and these are listed in the muParser section of the chapter on *Mathematical Expressions and Scripting*.

The first item is the *Curve Type* and is used to select the form of the generator function. The simplest and most common function definition is the classical Cartesian coordinate definition, y=f(x). This is the default option and is simply named "Function". Just below this is a text pane in which to enter the function definition, The next two parameters are used to select the X-range to be used for the plot. The last parameter sets the number of data points to be computed in the selected X-range. At the very bottom is a "Function Selector" which contains a dropdown list of intrinsic functions and brief description of the function. Clicking on the *Add Function* button will copy the selected function into the function definition pane.

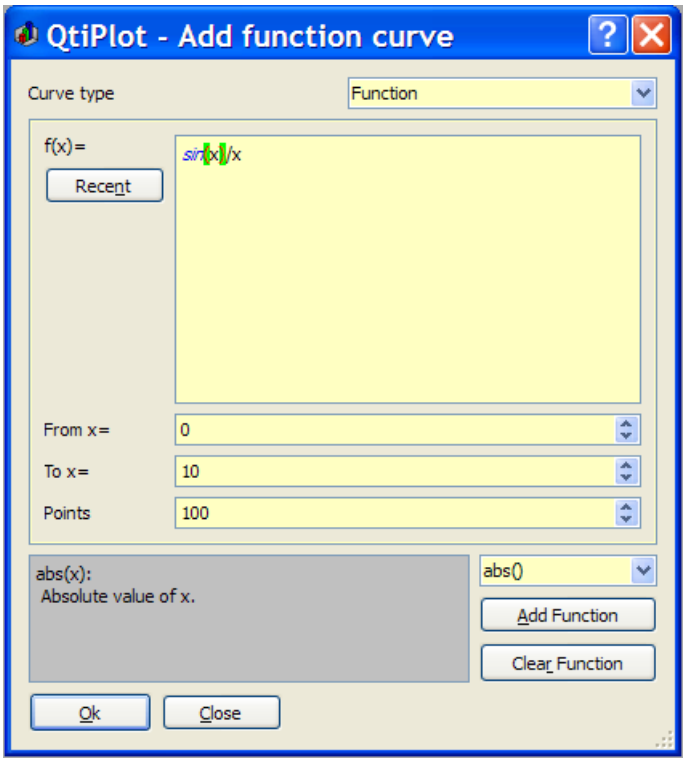


Figure 5.4: The **Add Function...** dialog box: Cartesian Coordinates.

If the function expression contains terms that are recognizable as constants, e.g.: f(x) = a*x + b, QtiPlot will detect them and display a two column table on the right side of the dialog which contains input spin boxes that simplify setting appropriate values for the detected constants.

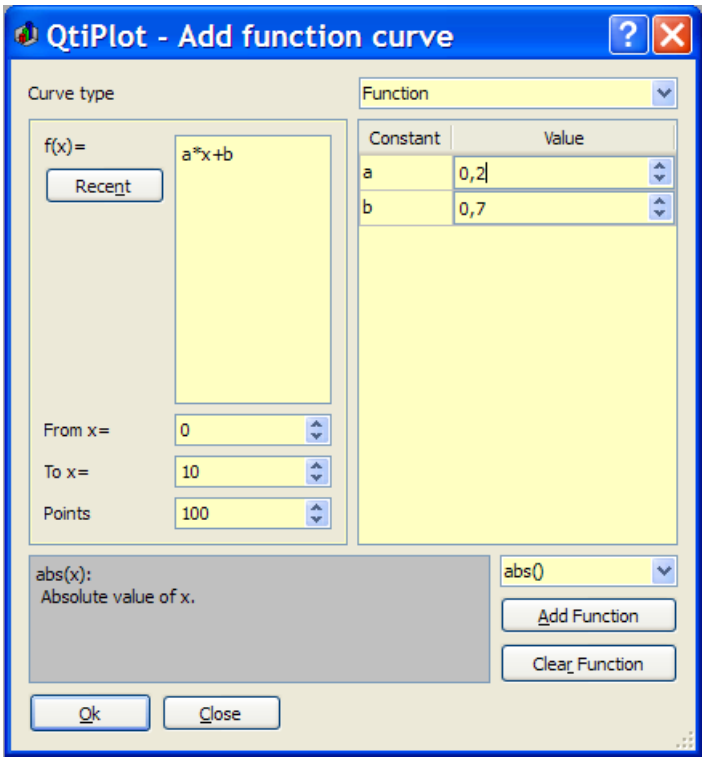


Figure 5.5: The **Add Function...** Dialog Box: Automatic Detection of Constants.

linkend=

A function can also be defined using a parametric definition by selecting "Parametric plot" as the *Curve type*. That is, given some variable, *m*, the (x,y) data points are computed using functions of that variable, x=f(m) and y=g(m).

The first parameter is the name of the parametric variable (here *m*). It is followed by the definitions of the two functions, the range of the parametric variable, and the number of data points.

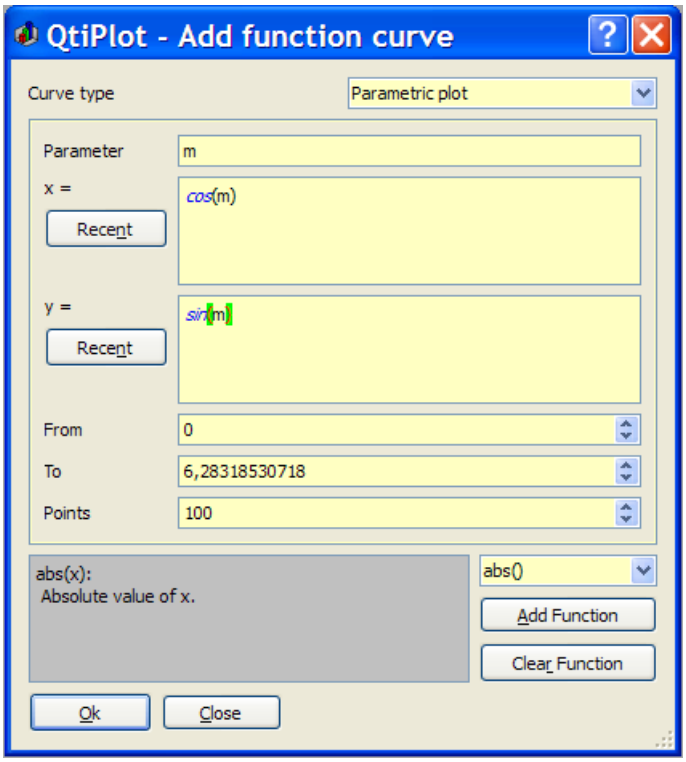


Figure 5.6: The **Add Function...** dialog box: Parametric Coordinates.

Finally, a polar definition of the function may be used by selecting "Polar plot" as the *Curve type*. Given some variable *t*, then the radius (*R*) and angle (*Theta*) are computed using two functions of that variable, R=f(t) and Theta=g(t), in a manner similar to the parametric function case. The (x,y) data points are then computed as x=R*cos(Theta) and y=R*sin(Theta).

The first parameter is the name of the parametric variable (here *t*). It is followed by the definition of the two functions, the range of the parametric variable, and the number of data points. Note that the angle is in radians. *pi* is an internally defined constant which can be used in any mathematical expression. For example, you can use *3*pi* to define the parameter range.

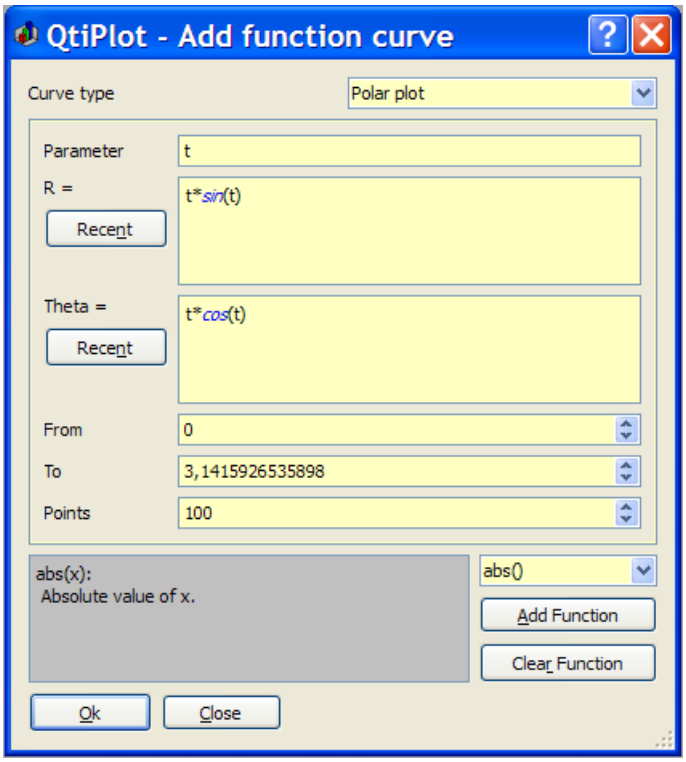


Figure 5.7: The **Add Function...** dialog box: Polar Coordinates.

5.4 Add Layer

This dialog is opened when you 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 and locate it on top of existing layers. The new layer contains an empty plotting area. You can subsequently modify the size and position of each layer by selecting it with the layer number buttons 1 2 and picking the *Layer Geometry* command from the context menu.

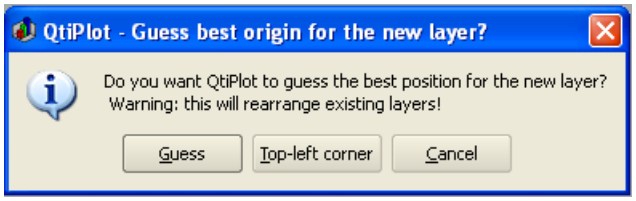


Figure 5.8: The **Add Layer** Dialog Box.

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 available from the project's tables that may be used for plotting. If the *Show current folder only* option is checked, only tables from the active folder of the project will be listed in this window. The right window lists the curves already plotted. In the case presented below, there are two tables that the **Add/Remove Curves...** dialog box lists for the selection of columns. You select a *Y* column to plot using this dialog. QtiPlot will use the column defined as X in the corresponding table for the *X* values.

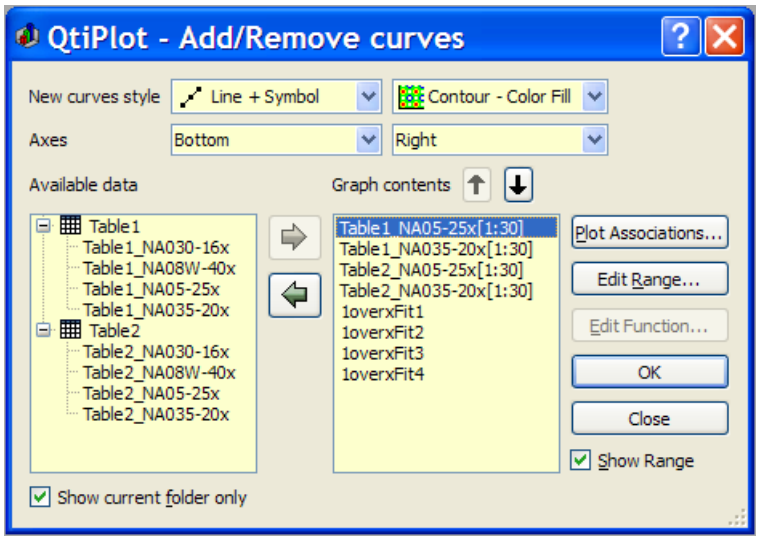


Figure 5.9: The **Add/Remove Curves...** Dialog Box.

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 from one table with the Y values from another one. If you want to do this, you have to copy the columns into 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 **Arrange Layers** command from the Graph Menuor by entering Shift-A.

This dialog provides for modification of the geometrical arrangement of plot layers which are already present in the active graph window. You can add new layers or remove existing ones using the *Number* of layers control. Checking the *Link X axes* option makes all the X-axes interdependent, meaning that if you manually set abscissa scale limits on one plot layer, then the same limits will automatically be set on all other layers in the plot window.

You can specify the number of rows and columns that define a grid of plots. With the default settings, QtiPlot computes the size of the layers in the grid from the size of the graph window. If you check the *Layer Canvas Size* option, you can set the size of the drawing area for the layers. QtiPlot will then modify the size of the graph window to accommodate them. When the *Fixed size* box is checked, the size of the layers will be kept unchanged when you manually resize the graph window.

The controls in the two group boxes on the right, *Alignment* and *Spacing*, are used to set the alignment of the layers in the window and the margins between the layer borders and the graph window limits. The *Spacing* group includes an *Align* control with 2 options (*Canvases* and *Layers*) that control the meaning of the row and column spacing entries. If the *Canvases* alignment option is chosen, then the spacing fields define the distances between the axes of the layers. On the other hand, if the *Layers* alignment option is chosen, then the spacing fields define the distances between the borders of the layers.

Finally, it is also possible swap two layers using the *Swap Layers* controls. Simply select the Source and Destination layers and click the *Swap* button.

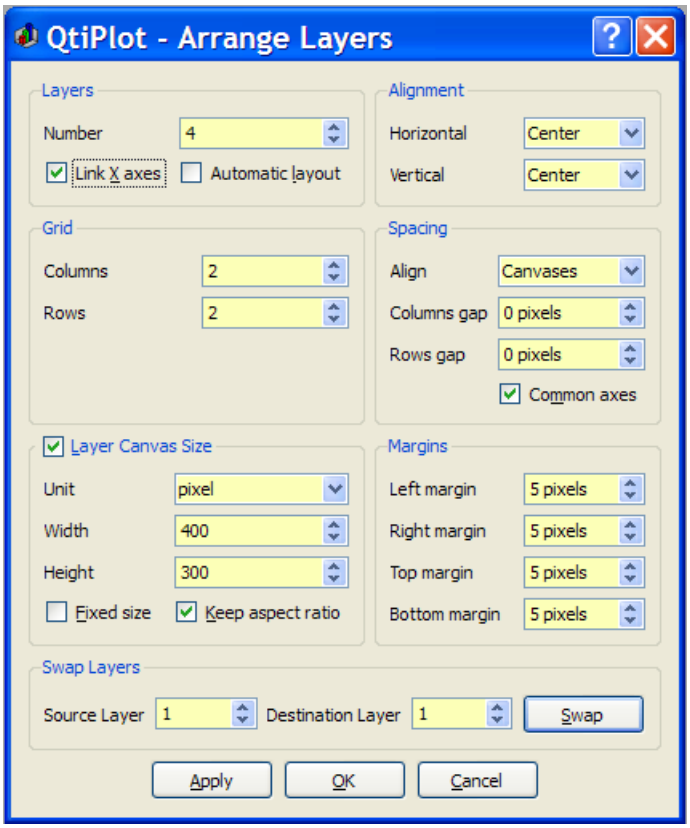


Figure 5.10: The **Arrange Layers** dialog: the Geometry Tab

Once suitable choices have been made, clicking on either the *Apply* or *OK* buttons will arrange the layers to obtain the desired alignment of the vertical and horizontal axis.

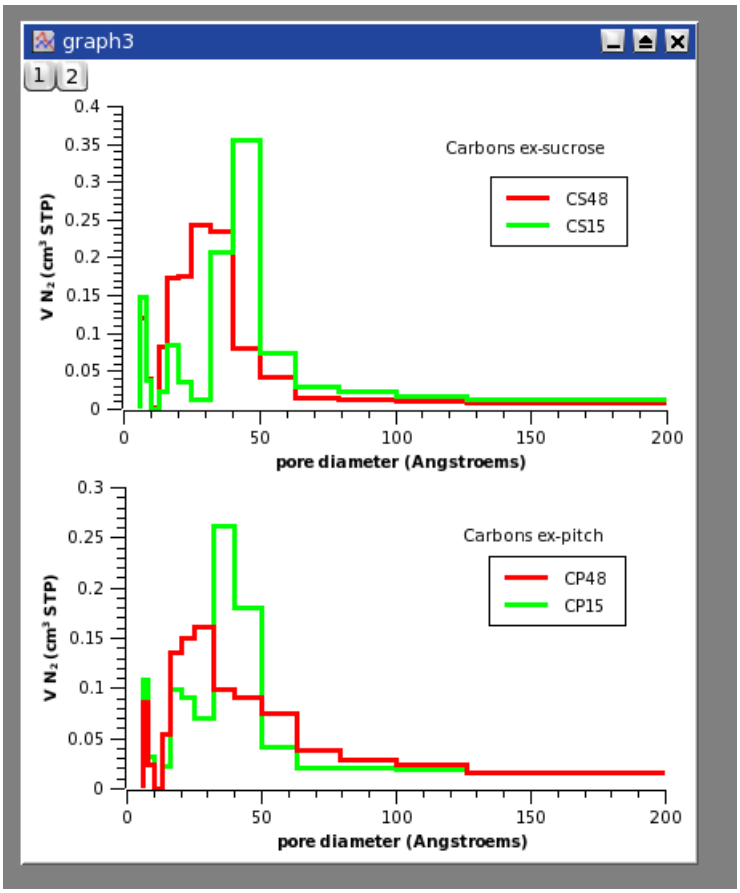


Figure 5.11: Example of a vertical arrangement for two plots.

Note that if modifications are later made to an aligned plots, then alignment of the different axes may be lost. Simply re-execute **Arrange Layers** to again re-arrange the plot.

5.7 Line Options

This dialog allows modification of lines or arrows which have been created with either the ***Draw Arrow*** command from the Graph Menu or by using Ctrl-Alt-A. The dialog can also be opened by double clicking on an arrow or line, or by right clicking a selected arrow or line and then selecting *Properties...* from the resulting pop-up menu.

The *Options* tab of the dialog allows changing the color, line type, line width, and the presence or absence of arrow heads. Line width is set in pixels. It is possible to define a default style for all new lines by clicking the *Set Default* button.

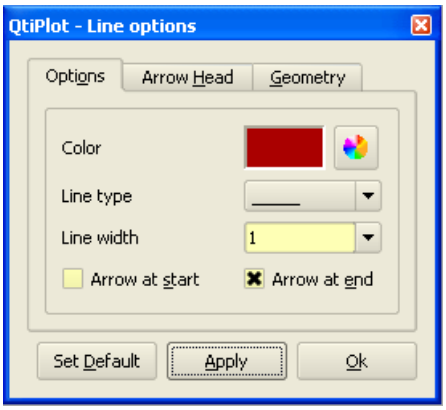


Figure 5.12: The *Arrow Options* Dialog: First Tab

The *Arrow Head* tab is used to modify arrow head shape. Length is in pixels and the angle is in degrees. It is also possible to define a default style for arrow heads using the *Set Default* button.

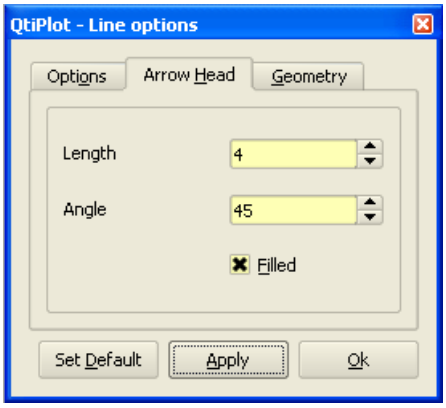


Figure 5.13: The *Arrow Options* Dialog: Second Tab

The *Geometry* tab is used to specify the starting and ending points of the selected line/arrow. The coordinates can be set as a function of the scale values displayed on the left (Y) and bottom (X) axes, or in pixels, by choosing the desired method from the *Unit* pull-down list. Pixel coordinates are relative to the top-left corner of the layer which contains the line.

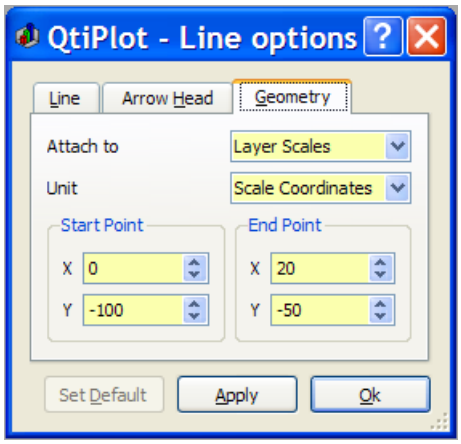


Figure 5.14: The *Geometry* Dialog: Third Tab

5.8 Column Options

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

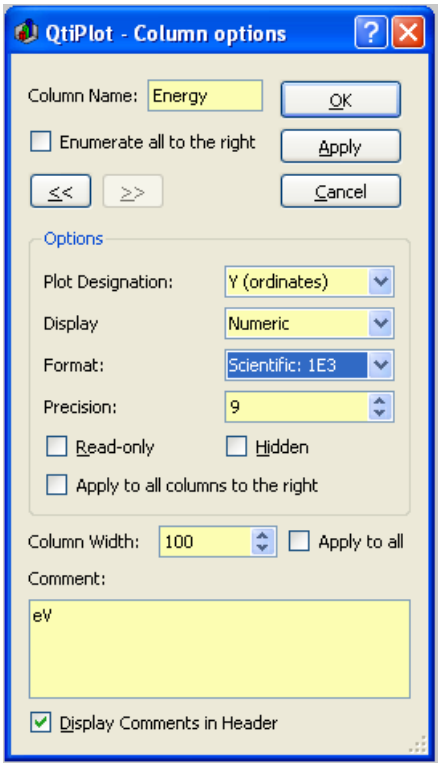


Figure 5.15: The **Column Options...** Dialog.

The checkbox *Enumerate all to the right* is used to rename of all the columns which lie to the right of the selected column. For example, if the name of the selected column is "xyz", then the selected column and all of the following columns will be renamed to "xyz1", "xyz2", and so on.

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

The *Plot Designation* selector is used to define which columns are to be used as X, Y or Z values or as error bars. In a table you can select several columns as X, in which case the column labels will display as [X1], [X2], etc. Multiple Y columns will display as [Y1], [Y2], etc. The same scheme is followed for multiple Z columns.

The *Comment* box allows entering text that will be displayed in the table header just below the column name when the *Display Comments in Header* option 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 from the Plot 3D menu: Contour+Color Fill command, Countour Lines command or Gray Scale Map command.

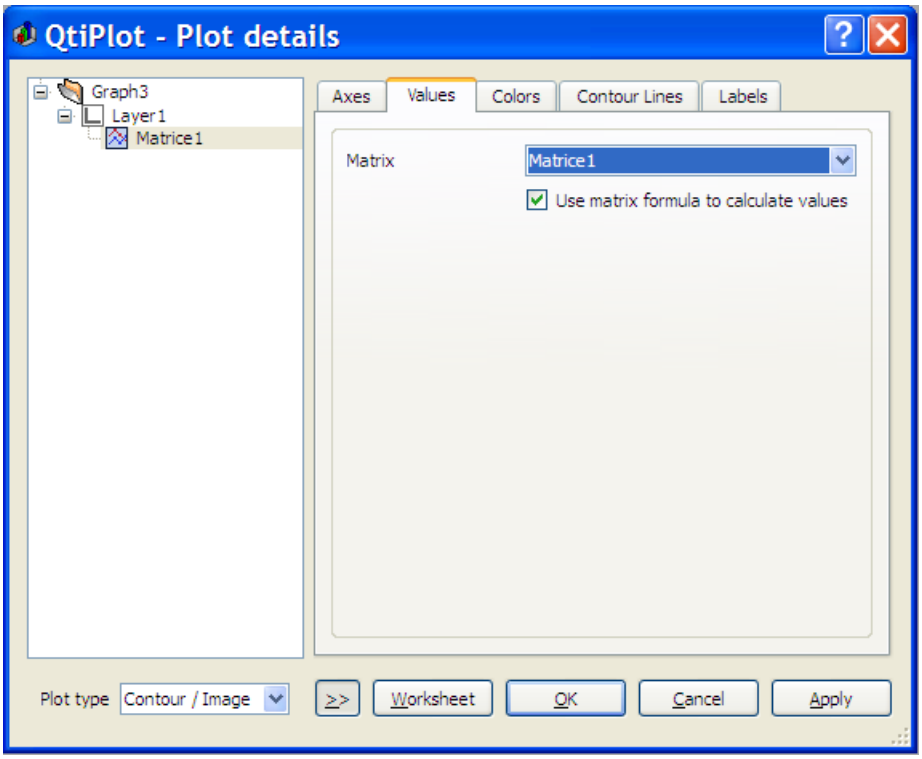


Figure 5.16: The Values tab.

The *Values* tab allows choosing the matrix that is to serve as the data source for the plot. It is possible to use a formula defined for the matrix to calculate the plotted Z values by checking the *Use matrix formula to calculate values* box. This results in higher accuracy, especially when drawing contour lines, but it only works if the formula uses muParser compatible syntax.

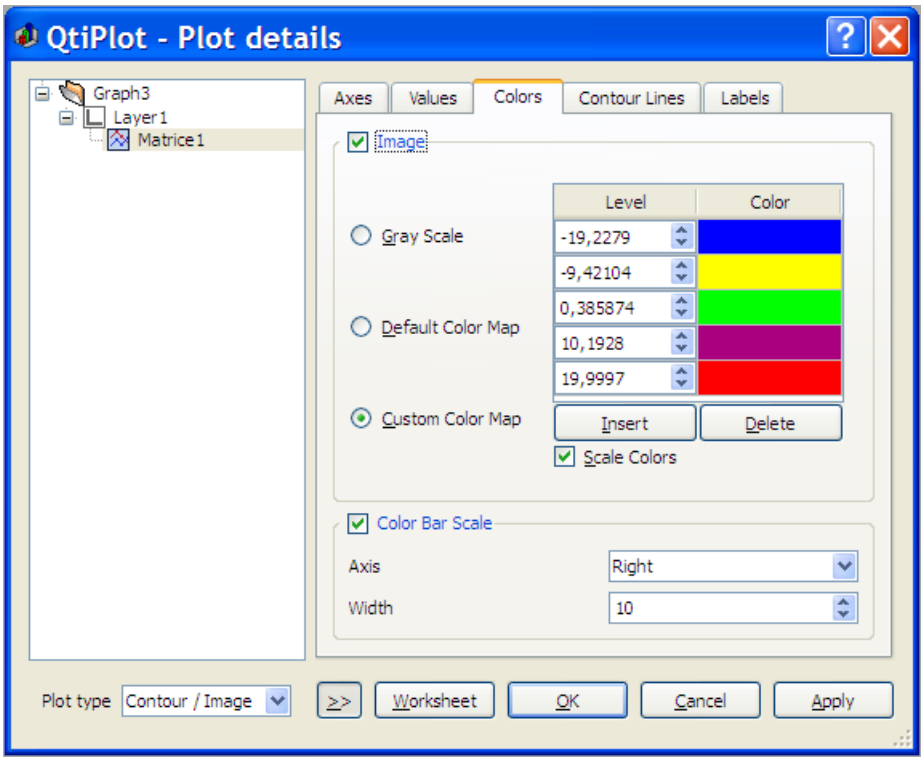
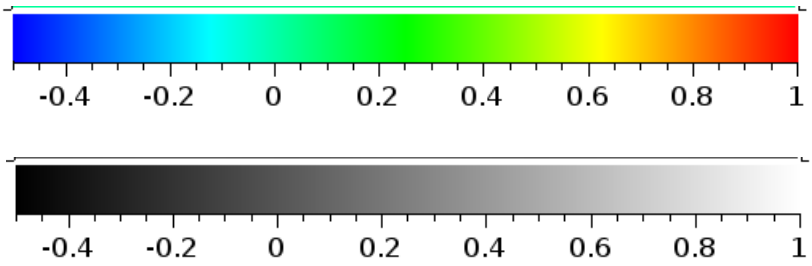
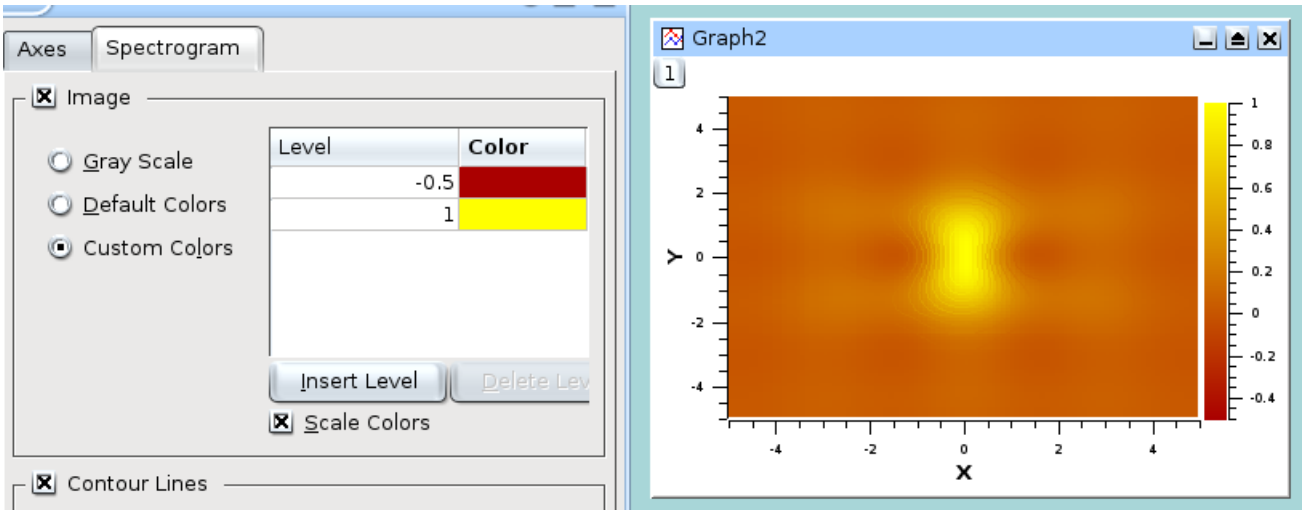


Figure 5.17: The Colors Tab.

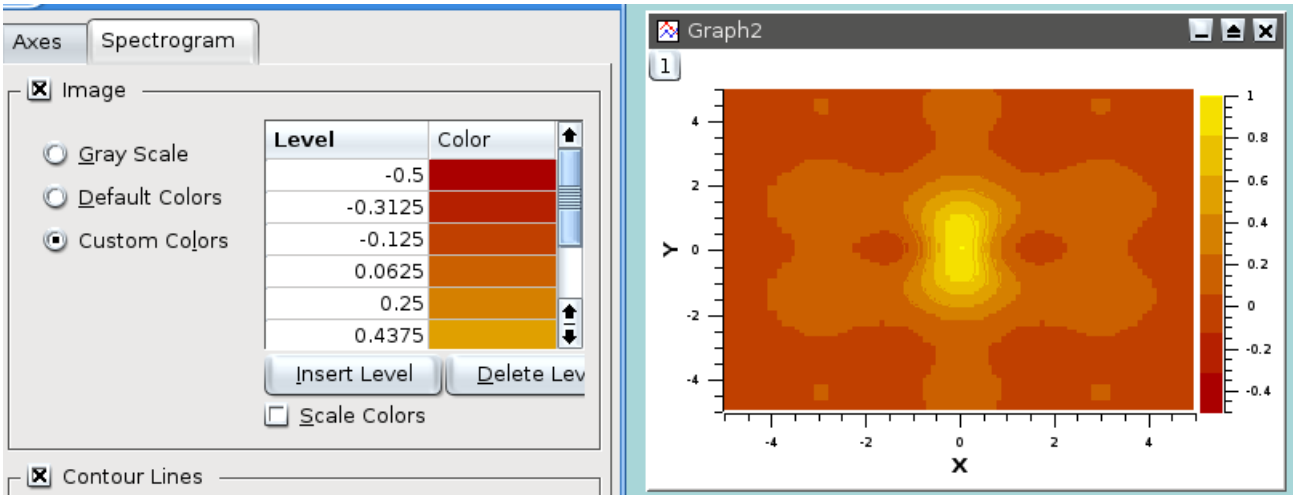
There are two groups of controls on this tab. Each group is activated by checking the box next to the group label. The first group of controls, labeled *Image*, are used to select the fill type of the contour plot. It is checked when filling of the contour plot is desired. If unchecked, no fill will be used. If filling is enabled, you may choose between gray-scale, a default color mapping and a custom color mapping with the 3 radio buttons. The default gray-scale and color maps are shown in the figure below.



You can only customize the colormap when the *Custom Colors* option is selected. A table with a set of numbers (the Z levels), and the color corresponding to each Z level, is presented. You can then add, remove or modify levels from the definition of the colormap. The first and last levels cannot be changed. These are always set to the minimum and maximum Z values. Note that this is only the definition of the colormap - it won't change the number of contour lines in your plot. An example of classical custom colormap is given here:



As long as the *Scale Colors* checkbox remains checked, Z-values that fall between colormap levels will be displayed in colors that are interpolated between the colors defined for the bounding levels. It is also possible to define discrete colors for each level. To do this you must uncheck the *Scale Colors* checkbox. In this case, no interpolation will be done, and you must define enough levels in your colormap to produce a desirable result.



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

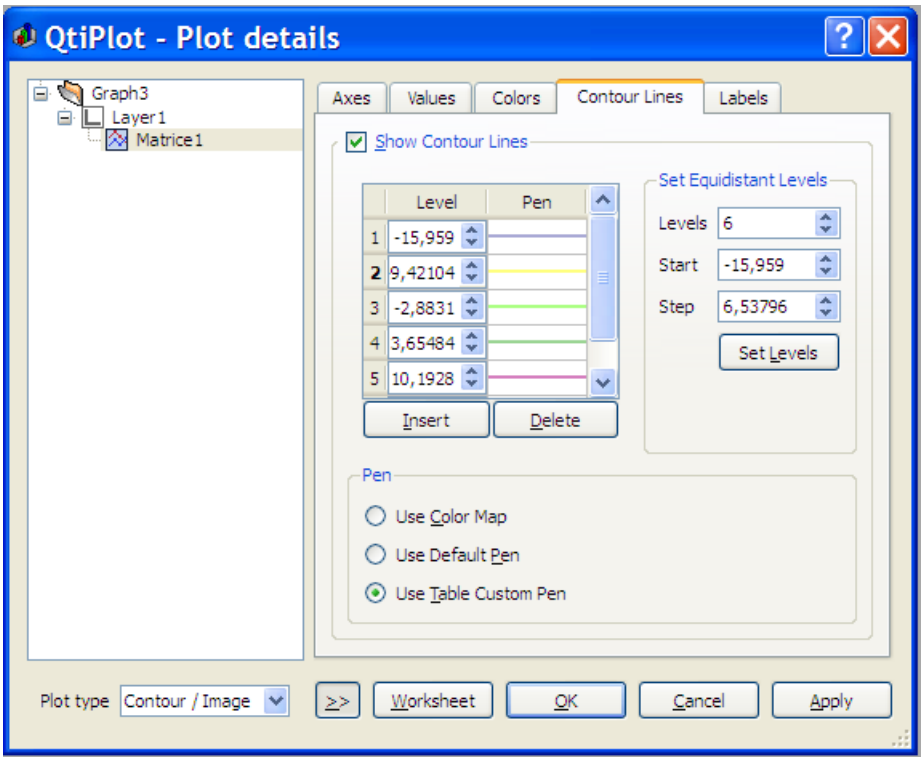


Figure 5.18: The Contour Lines tab.

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

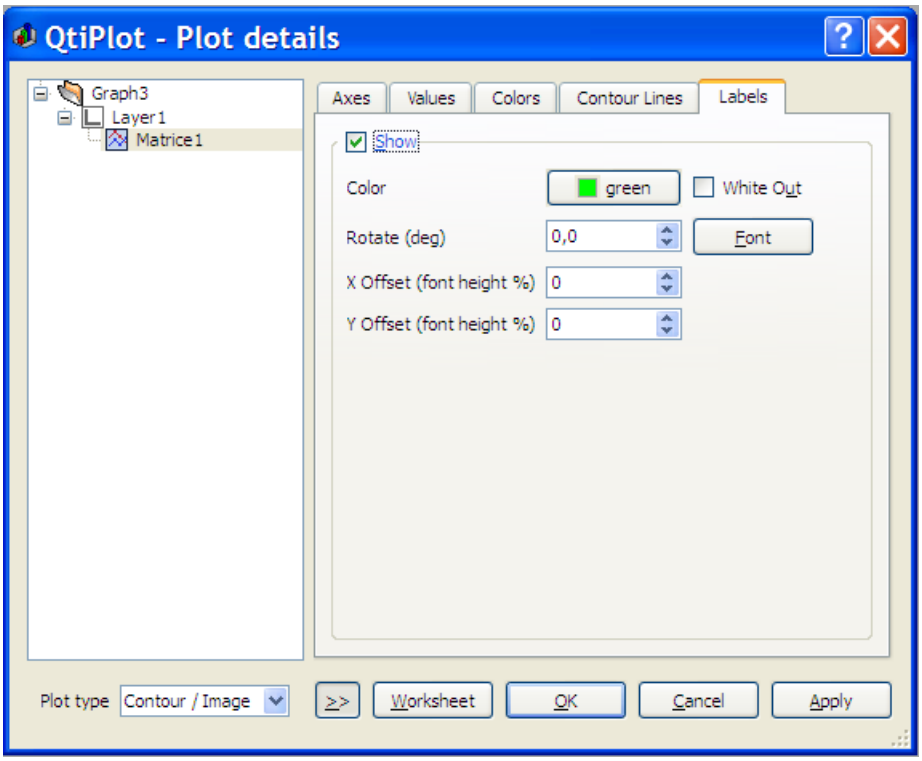


Figure 5.19: The Labels tab.

The *Labels* tab is used to enable/disable the display of a 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 Plot... from the Format Menu. It may also be activated either by a double click on a plot or by right-clicking on a plot and selecting *Plot Details...*. If there is more than one layer in the window, QtiPlot will select the layer which contains the plot under the mouse pointer. The exact contents of the dialog will vary depending upon how the dialog was invoked and on the type of plot selected.

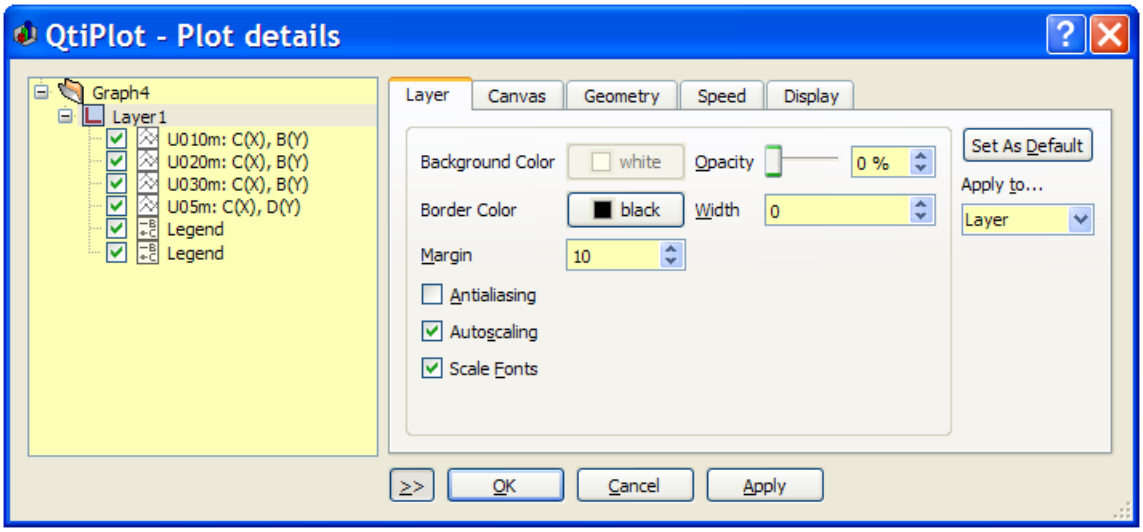


Figure 5.20: The Plot Details Dialog: Layer properties.

The second tab defines the properties of the plot canvas: the background color/image and the frame around the canvas.

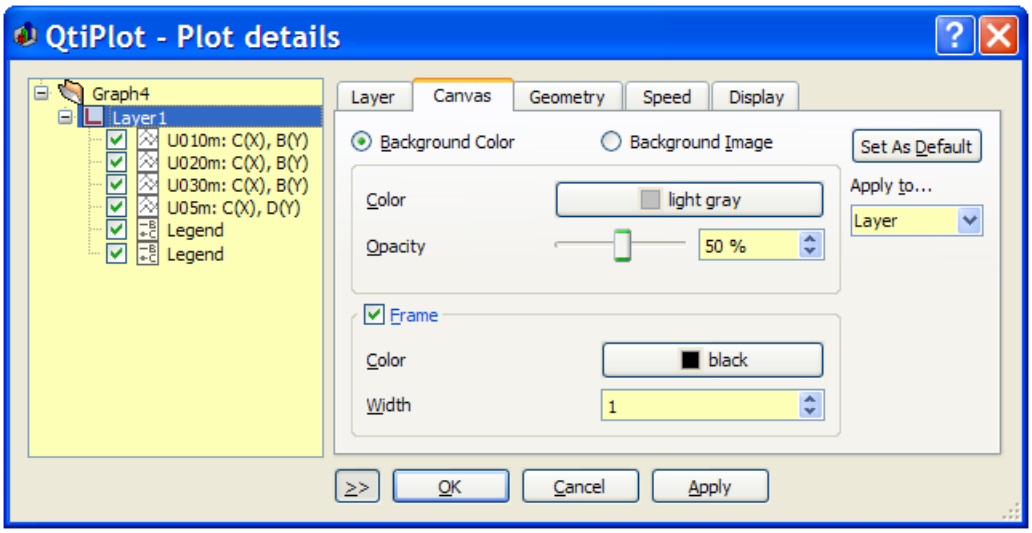


Figure 5.21: The Plot Details Dialog: Canvas with a solid background color.

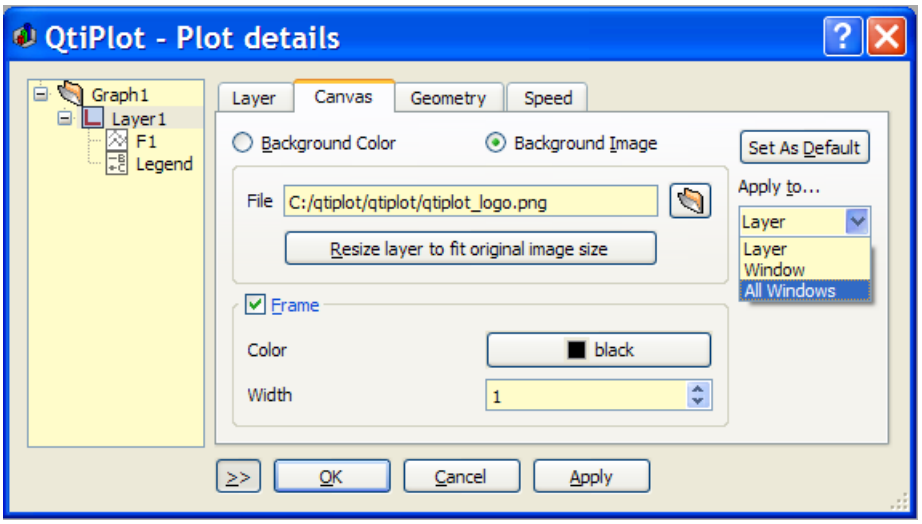


Figure 5.22: The Plot Details Dialog: Canvas with a background image.

The third tab defines the geometry of the plot layer.

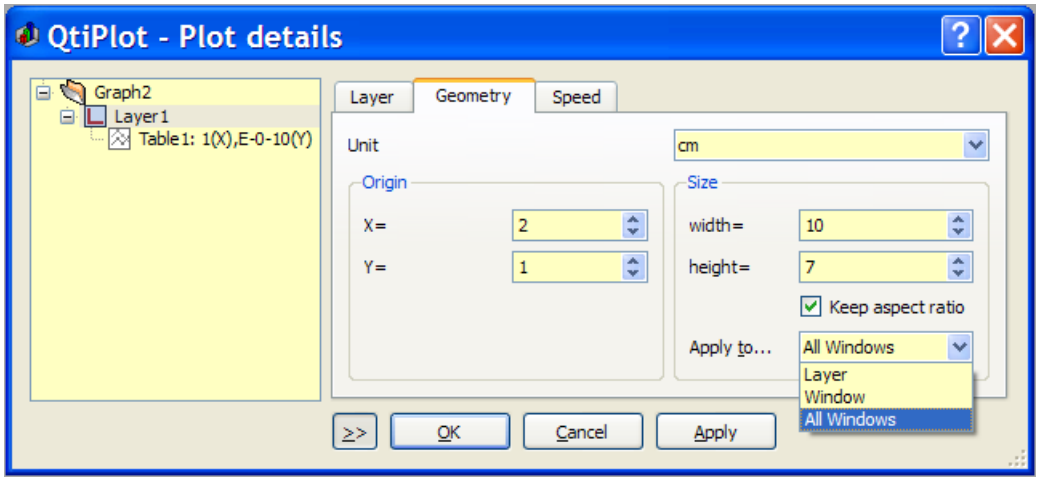


Figure 5.23: The Plot Details Dialog: Layer geometry.

The fourth tab can be used to customize the speed mode for the layer. This can be very useful when dealing with very large data sets. Speed mode uses the Douglas and Peucker algorithm, the purpose of which is to take a 'curve' composed of line segments (or points) and find a new curve with fewer points that is not too dissimilar from the original curve. The algorithm defines 'not too dissimilar' as a function of the maximum distance (tolerance) between the original curve and the new curve. The new curve then consists of a subset of the line segments (or points) that defined the original curve. When the speed mode is enabled, filtering of the data is activated only for the curves with more than the specified maximum number of data points.

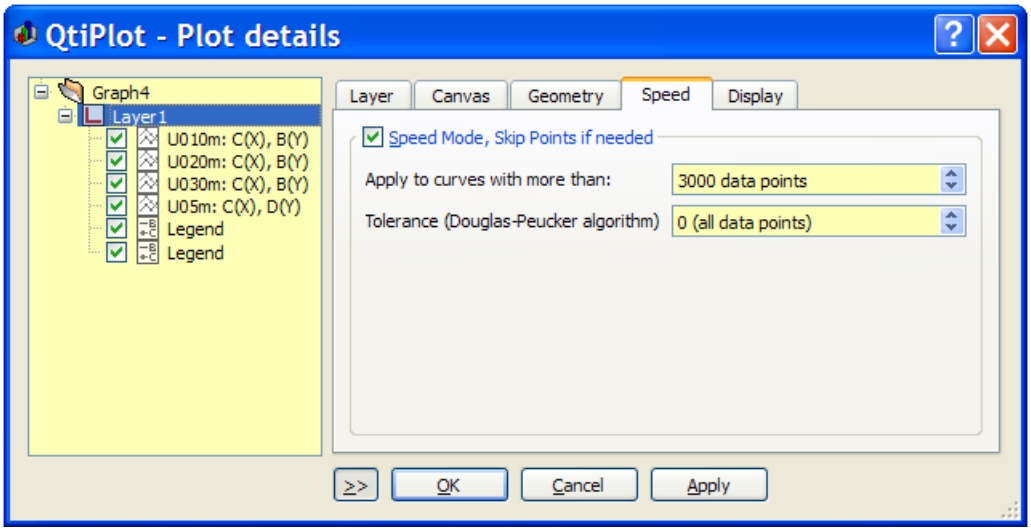


Figure 5.24: The Plot Details Dialog: Layer Speed Mode.

The right part of the dialog box contains several tabs which depend on the kind of plot selected. They are described in the following subsections. The left part of the dialog window shows the curves which are plotted on the active layer. All modifications will be made to the selected curve. If the *Plot Associations...* button is visible, then you can change which columns are used by clicking it. This will open a dialog which can be used to select the columns of the table to be used as X and Y values.

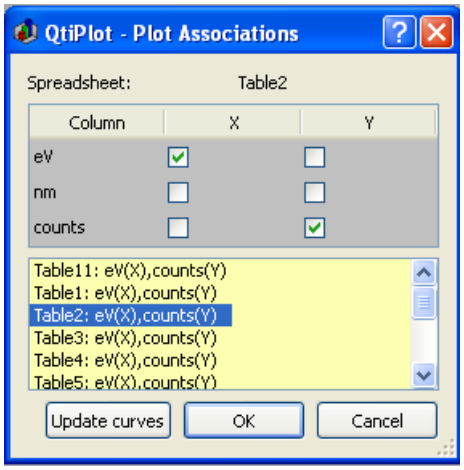


Figure 5.25: The Plot Details Dialog: Plot Associations.

If the *Worksheet* button visible, it may be used to access to the table which contains the columns for the selected curve.

5.10.1 Custom curves for lines and scatter plots

The first tab on the right side of the dialog window allows definition of the pair of axes to which a curve is attached. Attaching different curves on the layer to different X/Y axis pairs permits placing several curves with different x/y scales on the same plot layer.

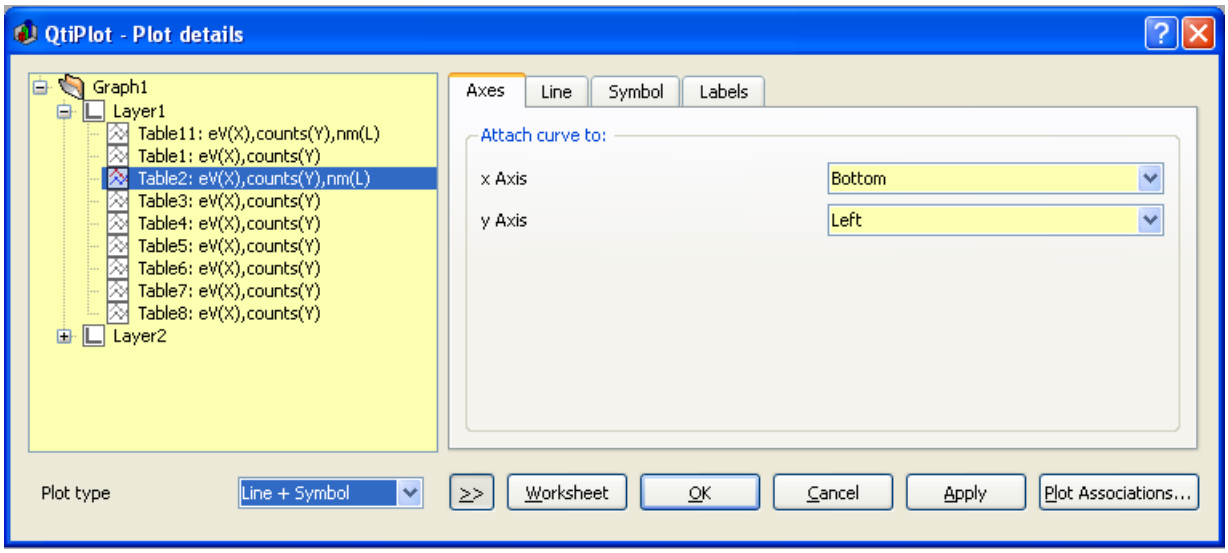


Figure 5.26: The Plot Details Dialog: Assign Axes.

The second tab on the right side of the dialog window allows customizing the style of the plot line (color, line style, thickness). The connect button allows changing the style which is used to draw the selected curve (steps, droplines, etc). See the Plot menu to see the different types of plots available.

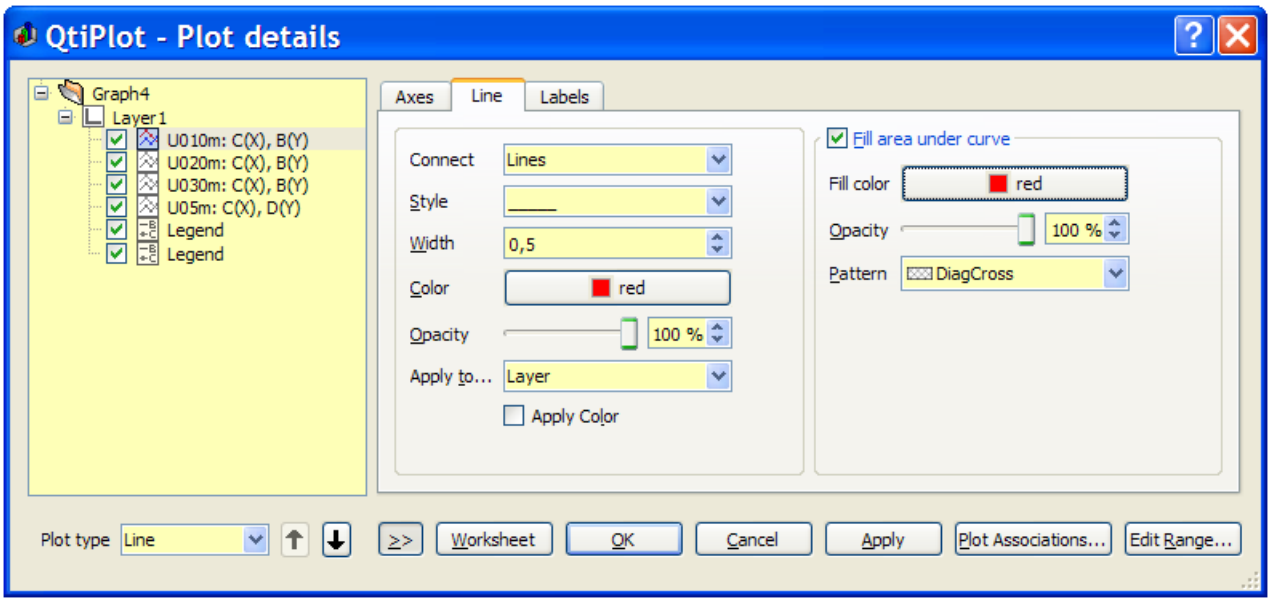


Figure 5.27: The Plot Details Dialog: Line formatting.

The third tab is activated to select the symbol, and to modify the size, color and fill color of the selected symbol.

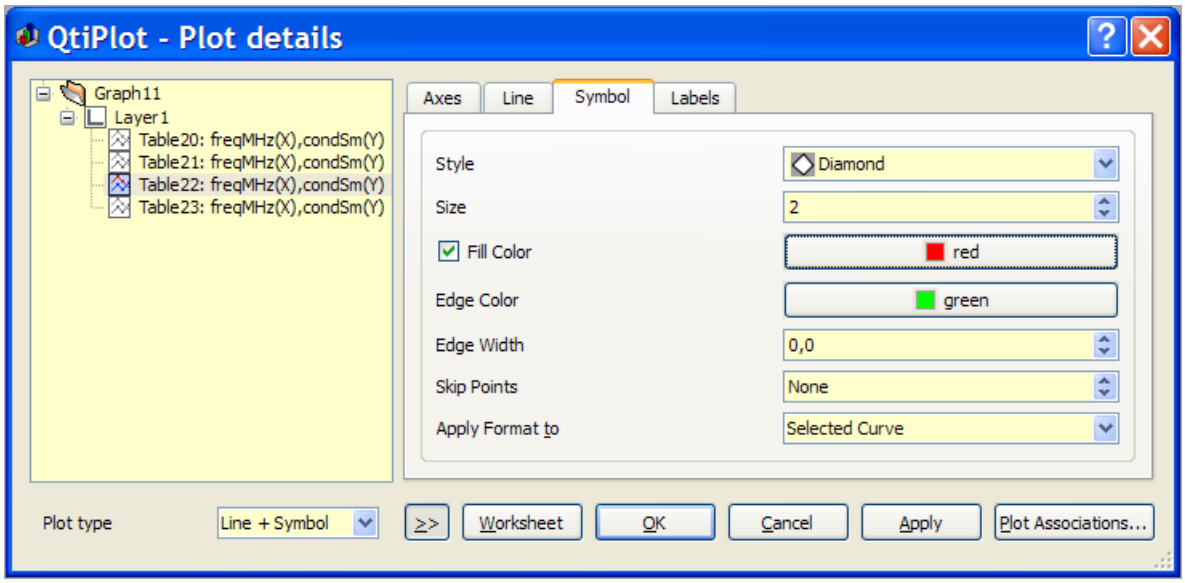


Figure 5.28: The Plot Details Dialog: Symbol formatting.

The *Labels* tab is used if you want labels plotted at each data point in the plot. Options are provided to control the font, color, rotation angle and position of these labels.

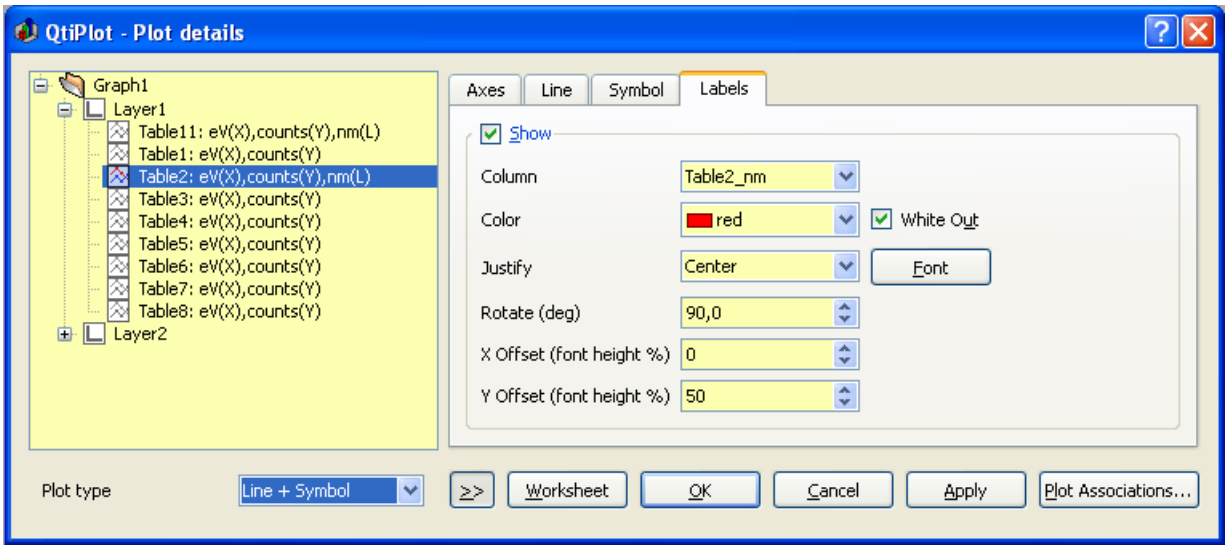


Figure 5.29: The Plot Details Dialog: Labels formatting.

5.10.2 Custom error bars

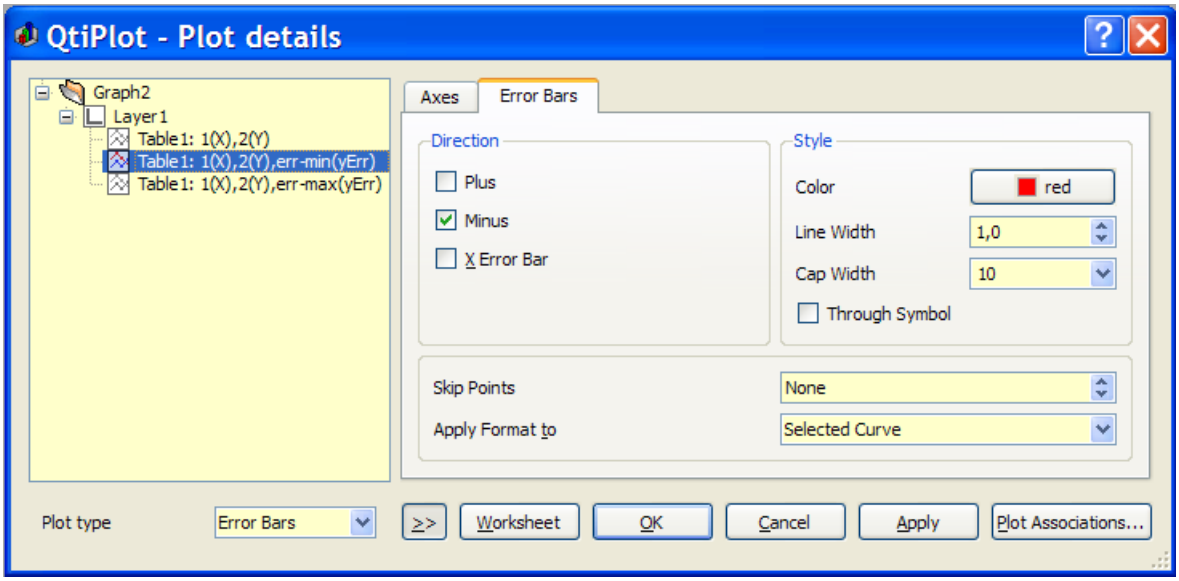


Figure 5.30: The Plot Details Dialog for formatting error bars.

5.10.3 Plot Details for pie plots

These commands are available for pie plots. The first tab is used to customize the pie segments. The fields on the left are used to modify the border drawn around each segment. The color, type and width of line may be customized. The default is no border (line width = 0).

The fields on the right are used to define the fill style used for pie segments. The *First color* combo-box defines which color is used for the first segment. The remaining segments will be filled with colors that follow the order defined in the combo-box's list. The default value for this field is black, so segments 2, 3, ... will be red, green, etc.

The *Pattern* combo-box defines the fill pattern that will be used for all segments of the pie. The default value is solid fill.

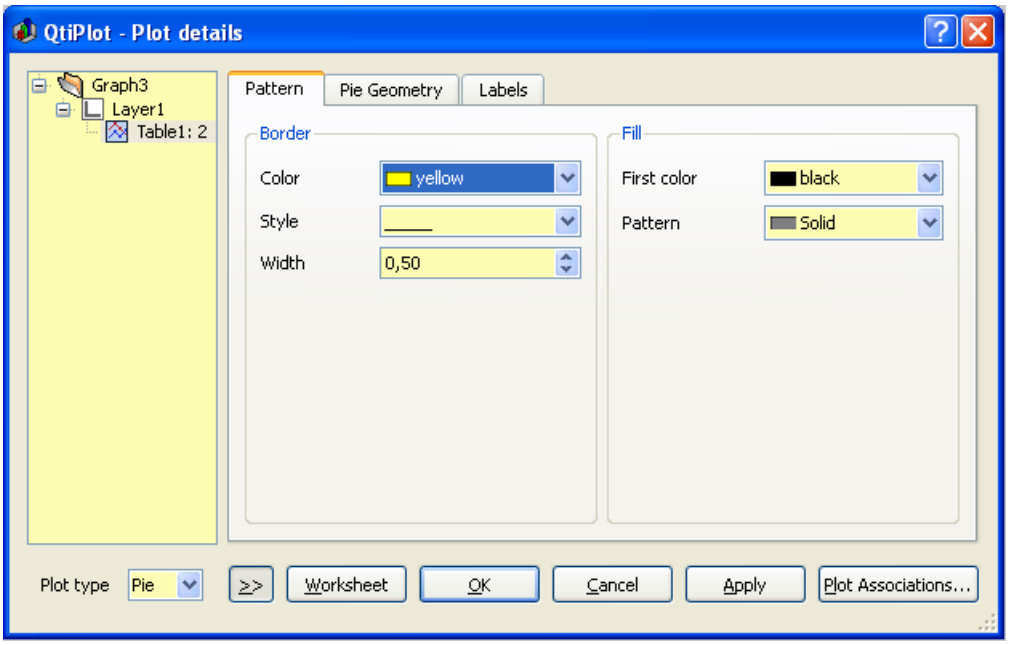


Figure 5.31: The Plot Details Dialog for pies: Pie Segment Formatting.

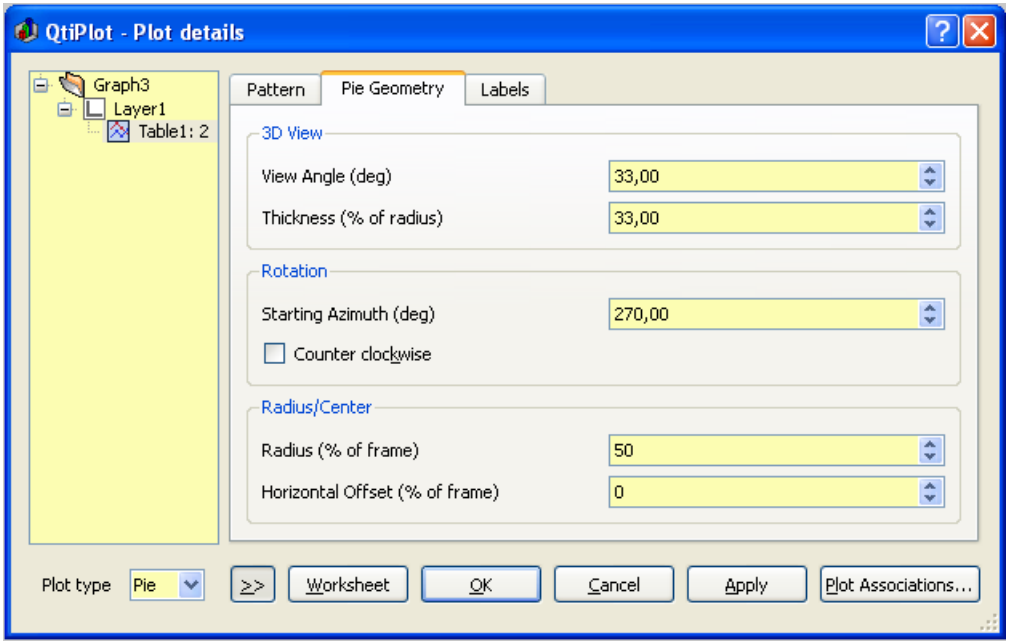


Figure 5.32: The Plot Details Dialog for pies: Pie Geometry.

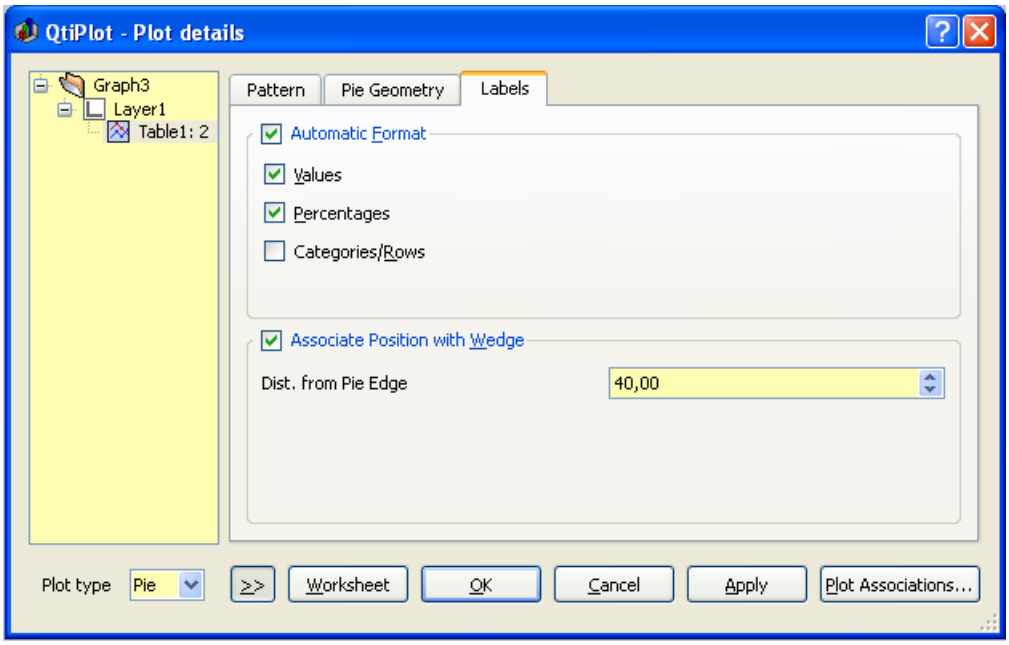


Figure 5.33: The Plot Details Dialog for pies: Pie Labels Formatting.

5.10.4 Custom curves for box plots

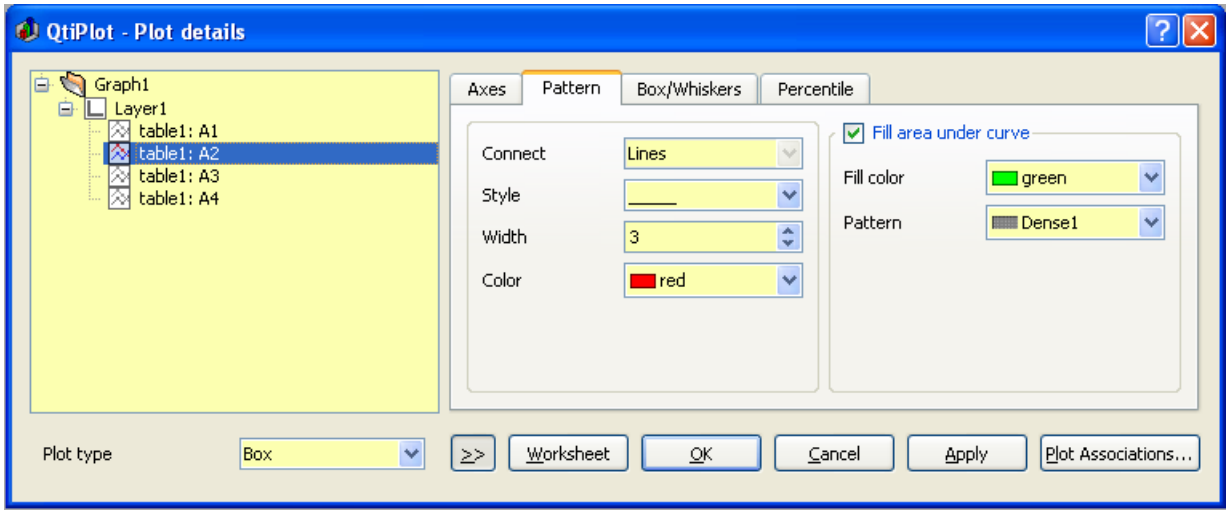


Figure 5.34: The Plot Details Dialog for box: Pattern Formatting.

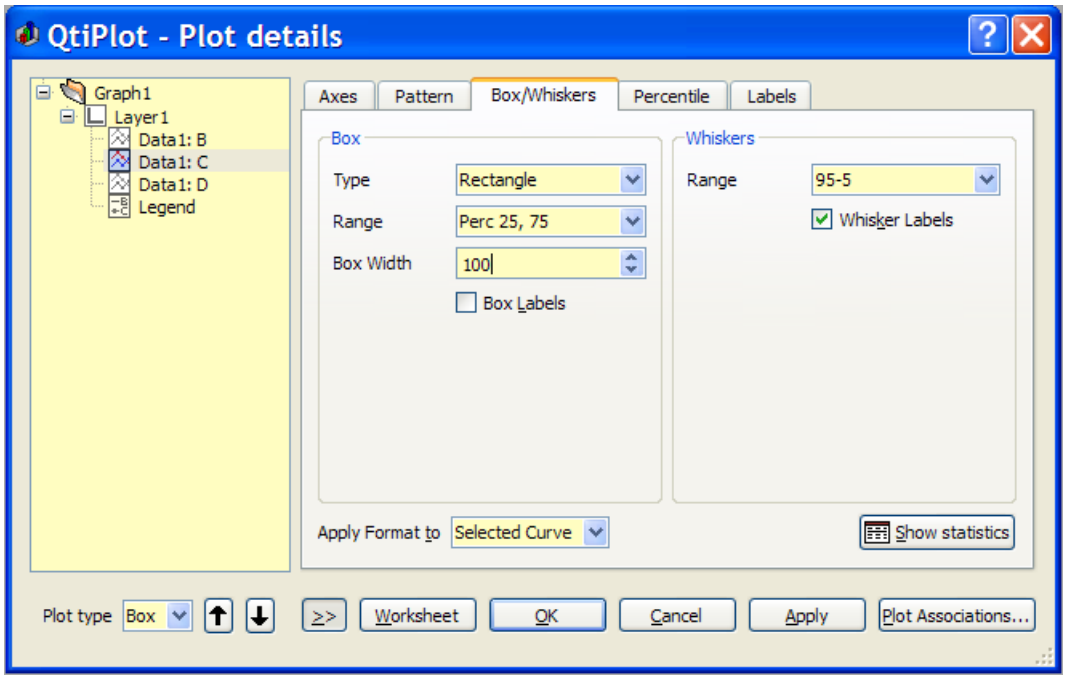


Figure 5.35: The Plot Details Dialog for box: Whiskers Formatting.

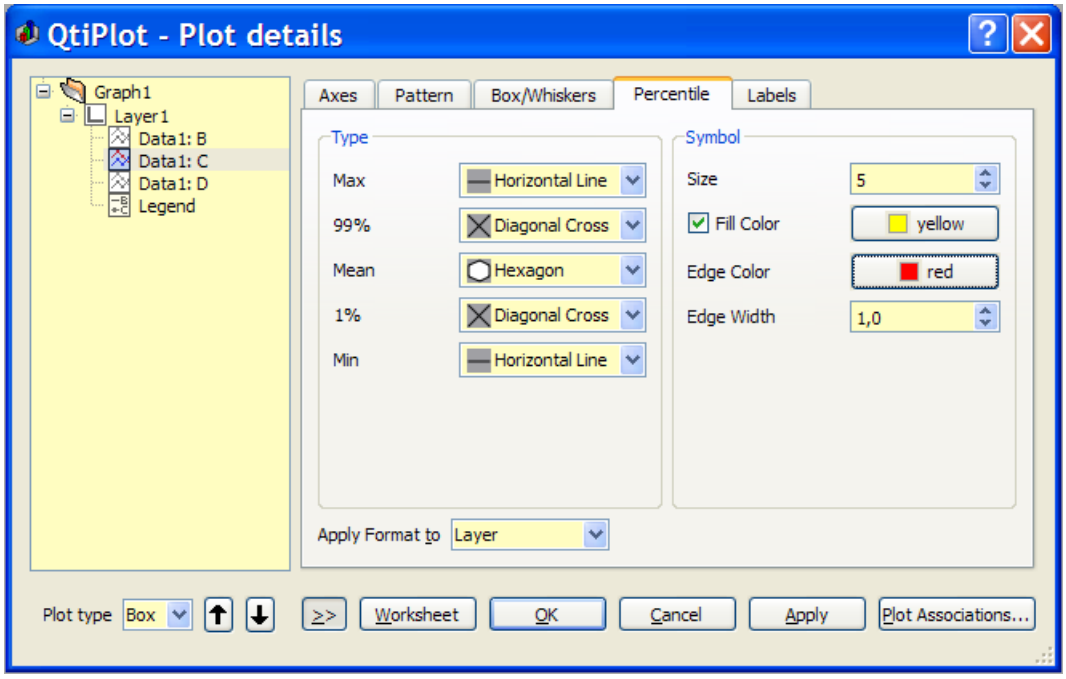


Figure 5.36: The Plot Details Dialog for box: Percentile Formatting.

5.10.5 Custom curves for pie histogram

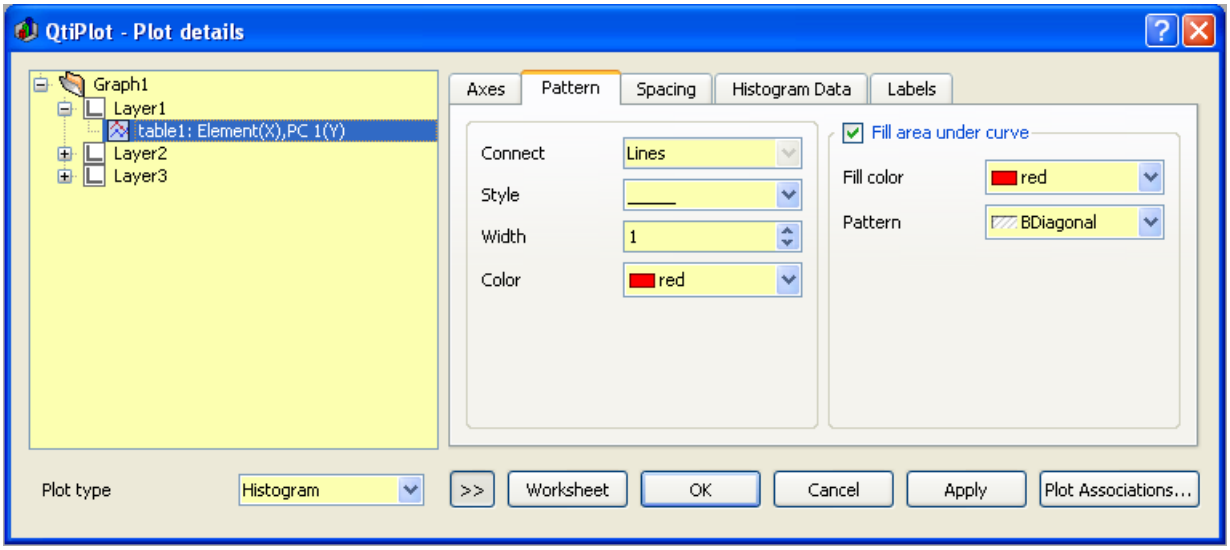


Figure 5.37: The Plot Details Dialog for histogram: Pattern Formatting.

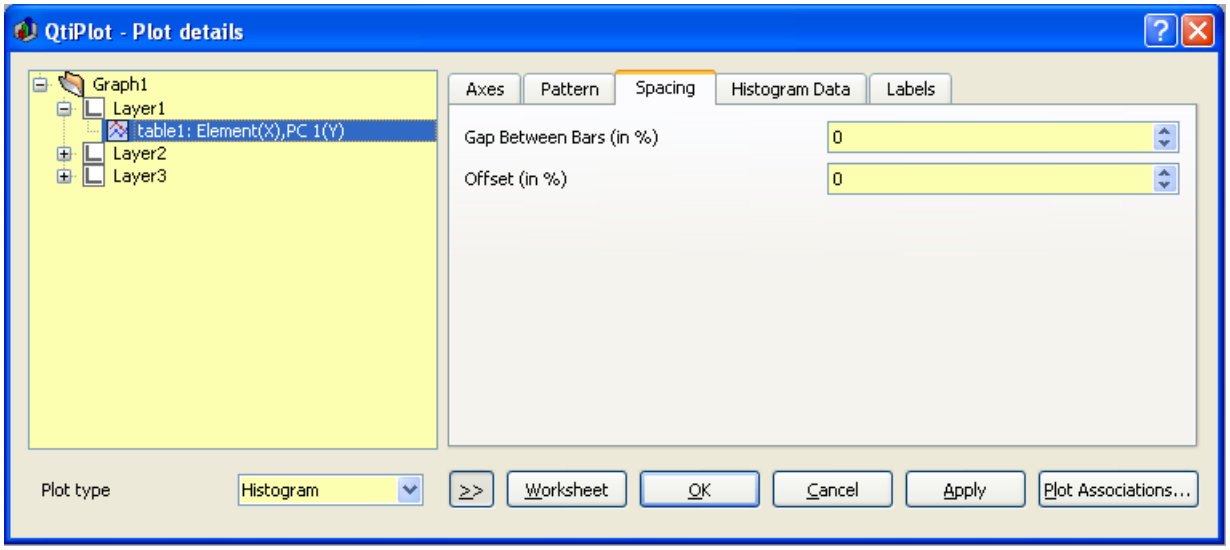


Figure 5.38: The Plot Details Dialog for histogram: Spacing Formatting.

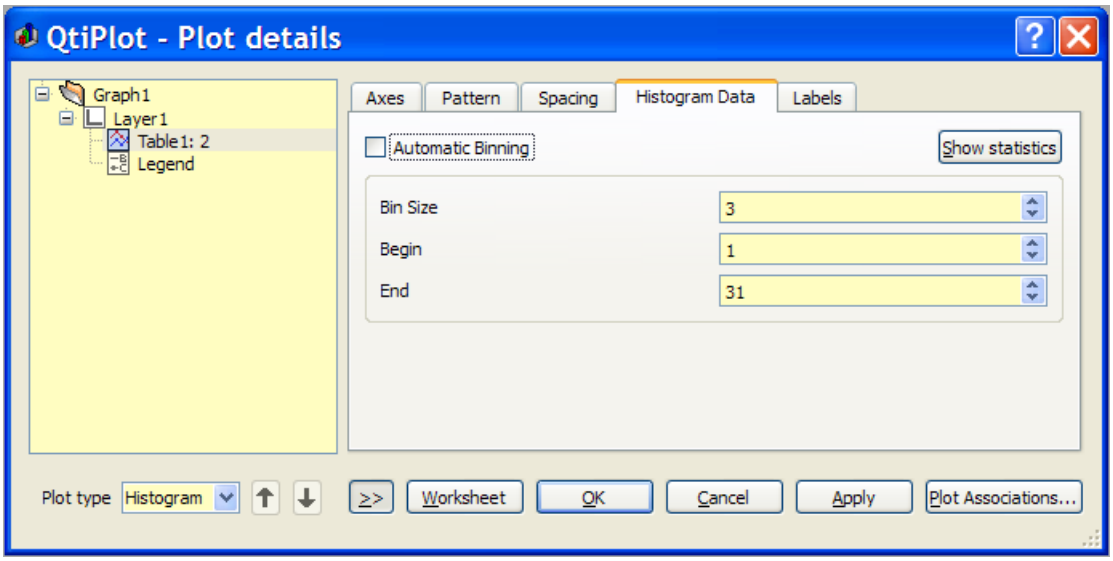


Figure 5.39: The Plot Details Dialog for histogram: Data Formatting.

5.11 Define surface plot

This dialog is used when you enter the **New -> New Surface 3D Plot** command. It allows for the creation of a new function of two variables. When the "Function" *Surface type* is selected the coordinate system will be Cartesian and the function will be of the form: z = f(x,y).

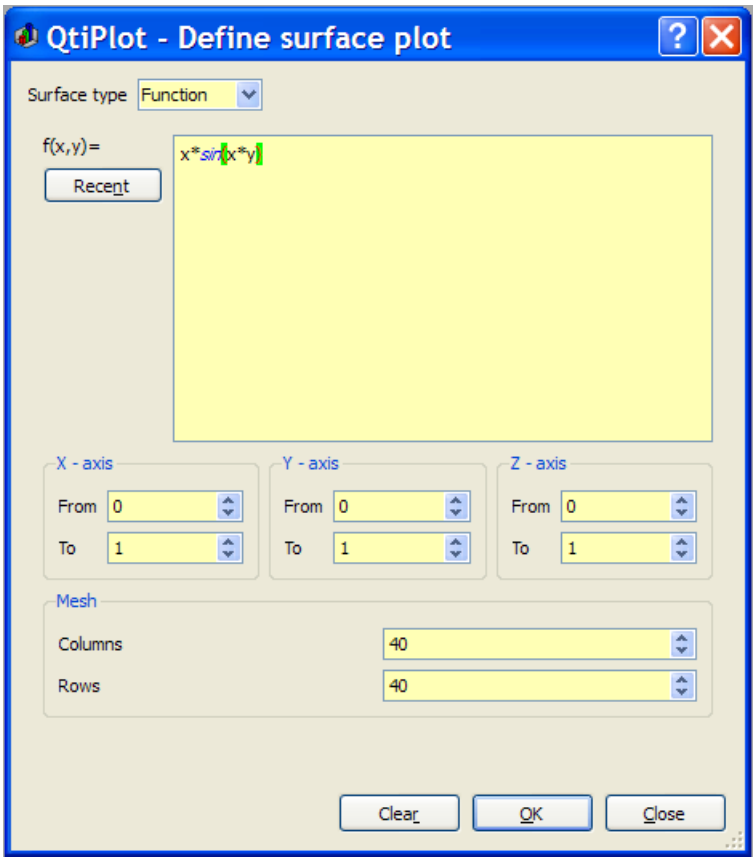


Figure 5.40: The **New -> New Surface 3D Plot** dialog box.

You may also define the X, Y and Z scales and the mesh parameters.

You can create parametric surfaces when the "Parametric" *Surface type* is selected. The only parameters allowed are latitude and longitude: `u` and `v`. Here, for example, is how you might plot a sphere having a radius equal to one:

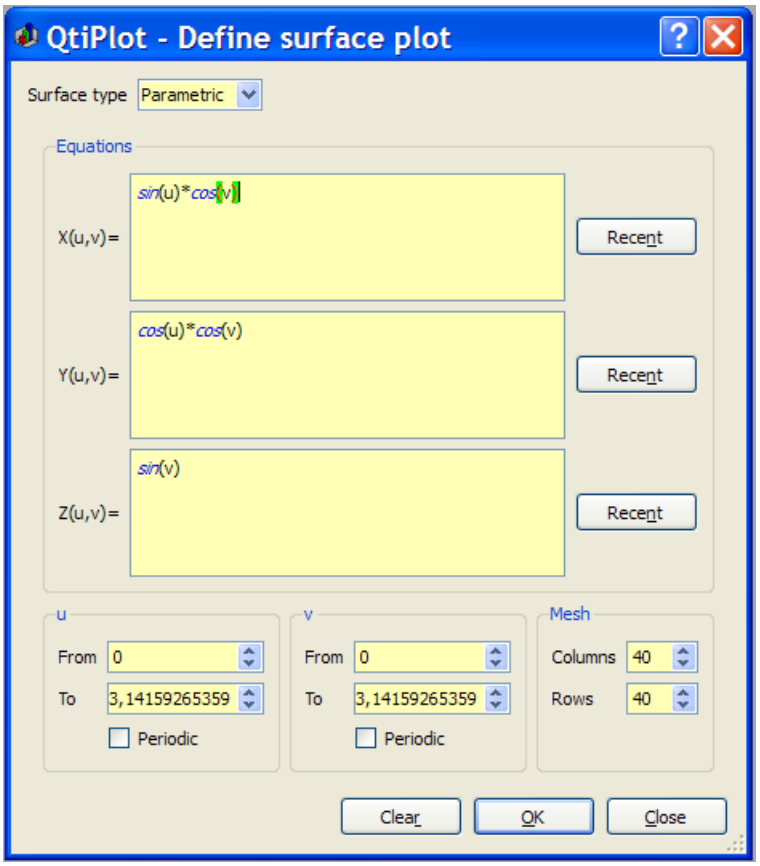


Figure 5.41: The **New -> New Surface 3D Plot** dialog box.

As in the case for functions, you can supply the drawing domain for the two angular parameters and you can define the mesh parameters. The more rows/columns you request, the better the resolution of the output image will be. However, this is at the cost of a larger memory consumption and an increased time of computation. A slow CPU and small amounts of memory will be limiting.

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

5.12 Export ASCII

This dialog is activated by selecting the Export ASCII command from the File Menu. It is active if there is at least one 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, the names of the selected columns will be exported as the first line of the file. If the *Include Column Comments* option were also checked then the comments displayed in the table header would also be exported as the second line of the file. A TAB character will be used as a separator between columns values. Since the *Export Selection* option is checked, only the selected columns will be exported.

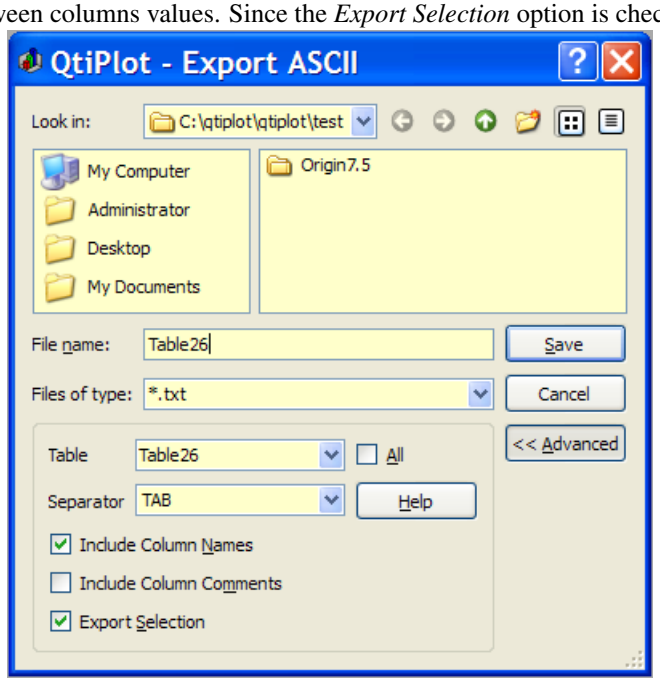


Figure 5.42: Export of a selection from a table to an ASCII file.

When the "Include Column Names" option is not checked, any column names will be ignored and the names "C1", "C2",... will be used in the exported file. The formatting of the numbers in each table column is preserved during the export. Make sure that adequate precision has been set before exporting to insure that the ASCII representation of the numbers in the output file is sufficiently accurate.

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 either the direct or inverse FFT. See the FFT section in the Analysis chapter for an example.

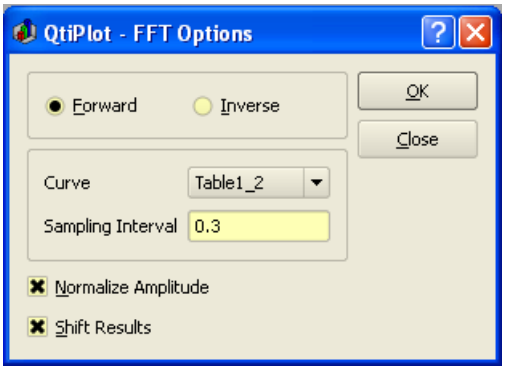


Figure 5.43: The **FFT...** dialog box for a curve.

QtiPlot will create a new plot window containing the FFT amplitude curve, along with a new table which contains the real part, the imaginary part, the amplitude, and the phase 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.

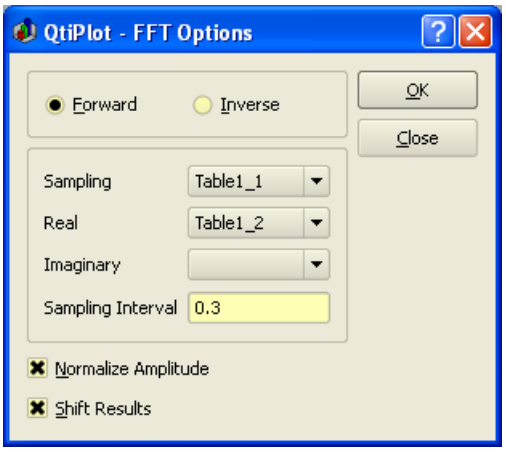


Figure 5.44: The **FFT...** dialog box for a table.

When performing an FFT on a table, you must select the sampling column (X-values) and one (for simple real values) or two (for complex values) columns of Y-values. Complex numbers have the real part of the Y-values in the first column and the imaginary part of the Y-values in the second column. If Y-values are simple reals, you don't have to select a column for the imaginary part. You can leave the combo box for the second column empty in this case.

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 fewer values.

5.14 Integrate Function Dialog

This dialog box is opened when you select the **Integrate Function...** command from the Analysis menu Pressing the *Integrate* button performs a numerical integration of a user defined analytical function. This function is entered/edited in the *Function* text box. A combo box which lists internally defined functions is provided. Clicking the "Add" button will copy the selected function into the *Function* text box at the position of the text cursor. Clicking the "Clear" button will clear the entire contents of the *Function* text box. Clicking on the "Recent" button opens a dialog which contains a combo box that lists recently used functions. Selecting one of these and clicking "OK" will replace whatever is in the *Function* text box with the selected function. The second input field displays the name of the independent variable of the function (i.e., the variable of integration). It is set to *X* by default. The third parameter is the maximum number of subintervals that that the integration algorithm may use to integrate the function. The default value of 1000 should work in almost all circumstances. The fourth parameter is the tolerance limit. Smaller numbers will result in the algorithm using more subintervals. The default value of 0.01 should work for most functions. The next two fields are the lower and upper limits of integration. Set these to whatever values you need. The final item is the *Plot Area* check box. Leave this checked if you want a plot of the integrated function to be drawn on the currently active layer. A table of the plotted data can be generated by right clicking on the curve and selecting the *Worksheet* command.

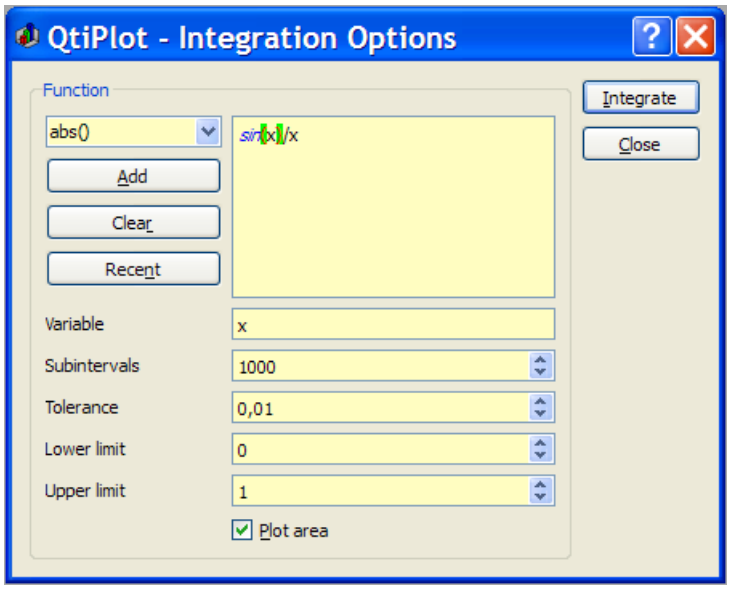


Figure 5.45: The **Integrate Function...** dialog box.

The numerical result of the integration will be listed in the Log Panel.

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 select one of 4 *Categories* of functions: 1) user defined functions which have been previously saved, 2) the classical functions provided by QtiPlot in the analysis menu, 3) simple elementary (basic) built-in functions, and 4) external functions provided via plugins.

To choose a function, first select a category and then the desired function from the displayed Function list. Clicking on the checkbox under the selector will clear the contents of the function entry text pane (see below) and copy the selected function into it. You can also click the *Add expression* button to copy the selected function, but this will not clear any previous contents. If you've selected one of the "basic" functions, there will be no checkbox and you will need to use the *Add expression* button.

The bottom half of the dialog box allows you to define your own function. You can either write you own mathematical expression from scratch or add expressions from the function selector with the *Add expression* button. Once a custom expression is completed, clicking on the *Save* button will add the function to the list of user defined functions. The *Name* field will be used as the name of the function. A copy of the function is saved on disk with the extension ".fit". You can define the folder where ".fit" files are saved using either the *Choose models folder...* button (shown only when "User defined" is selected) or by selecting a new folder in the Save file dialog. Functions can be removed from the User Defined list by selecting them and clicking on the *Remove* button. You will be asked to confirm the deletion.

This first step is used to define the function which will be used for fitting. When you are ready to perform the fit, click on the button to activate the *Fitting Session*.

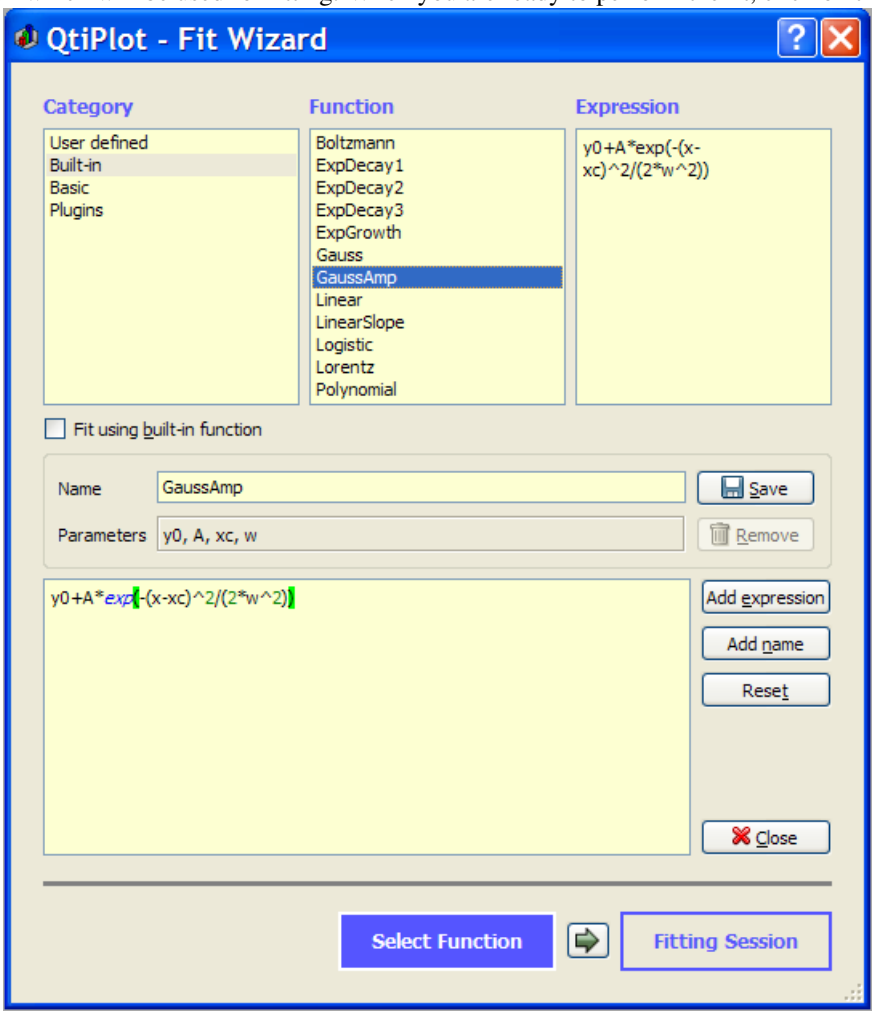


Figure 5.46: The first step of the **Fit Wizard...** dialog box.

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

This second step is used to define the parameters of the fit.

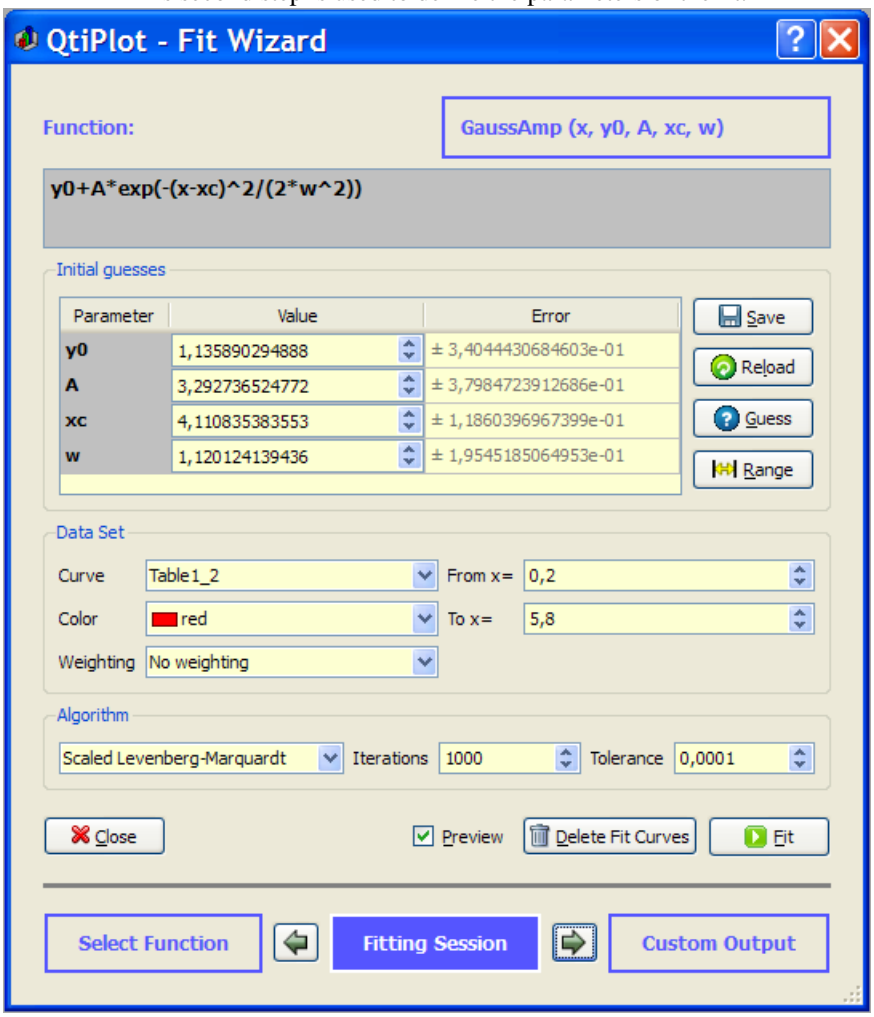


Figure 5.47: The second step of the **Fit Wizard...** dialog box.

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 coefficients are set to 1 ($w_i = 1$).
2. *Instrumental*: the values of the associated error bars are used as weighting coefficients $w_i = 1/er_i^2$, where er_i are the error bar sizes stored in error bar columns. You must add Y-error bars to the analyzed curve before performing the fit.
3. *Statistical*: the weighting coefficients are calculated as $w_i = 1/y_i$, where y_i are the y values in the fitted data set.
4. *Arbitrary Dataset*: allows setting the weighting coefficients 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*: allows setting the weighting coefficients 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. It can be found in and viewed from the project explorer.

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

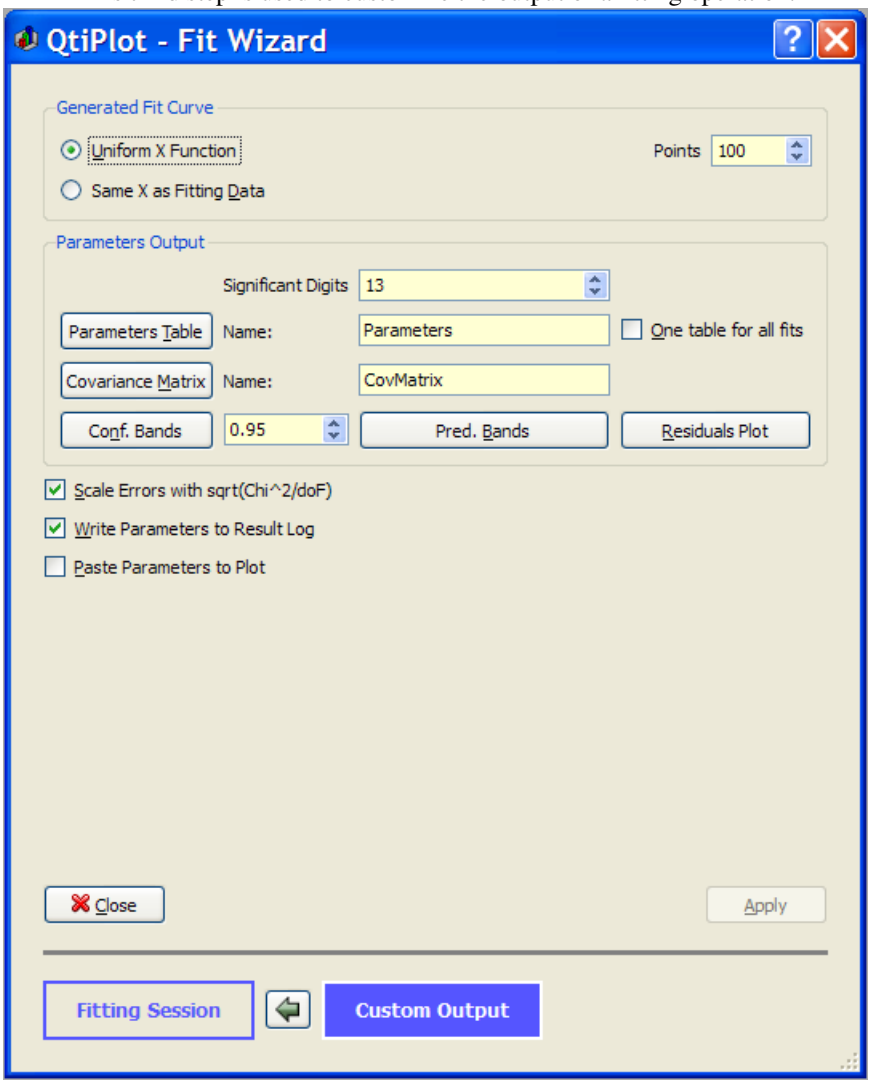


Figure 5.48: The third step of the **Fit Wizard...** dialog box.

This dialog tab provides controls that can be used to evaluate the goodness of fit. The first is the *Residuals Plot* button which is 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 evaluation of the goodness of fit. These 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, reported errors are not automatically scaled by 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 two or three axis plots.

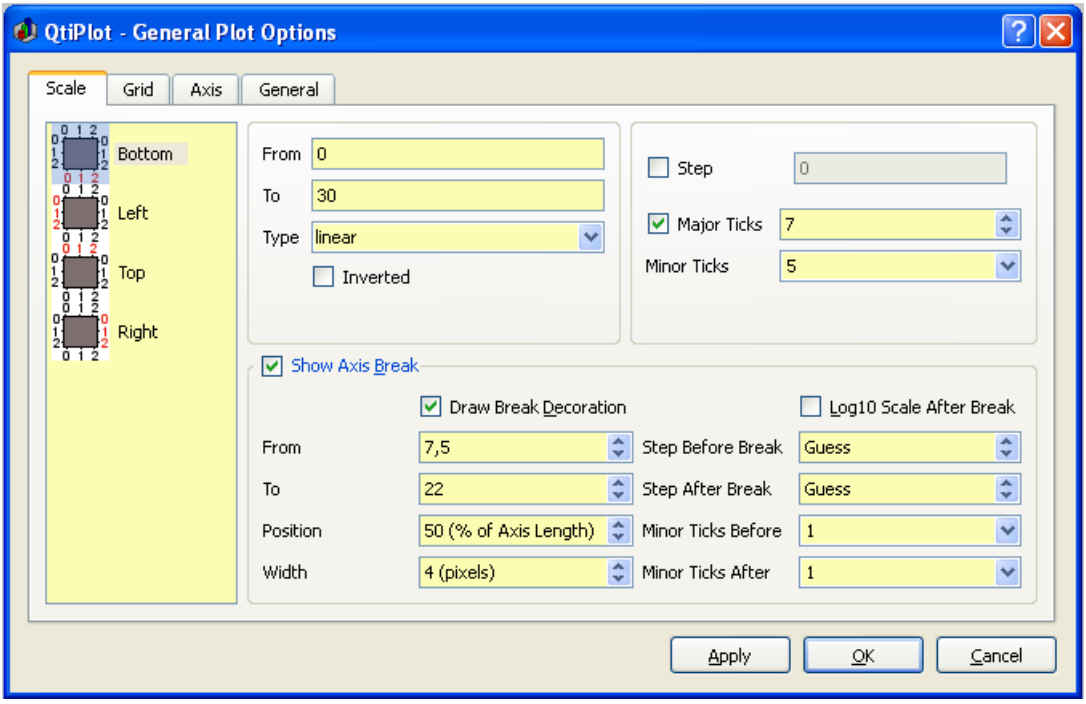


Figure 5.49: General plot options dialog: The Scale Tab.

In this tab, you can set the number of ticks used for each axis. This can be done in two ways: 1) Set the number of labels to use for the entire scale. Whatever number you enter, QtiPlot may adjust the value so as to produce a prettier plot (for example, if you enter 7 ticks for a 0..100 scale, QtiPlot will use 10 major ticks spaced evenly from 10 to 100). 2) Set the Step Increment and QtiPlot will generate as many ticks as needed to fill the scale range using the indicated increment. This produces tick spacings and labels which are more like those that are typically done on hand drawn graphs.

The grid tab is used to draw grid lines on the plot. The frequency of the lines are related to the number of labels and major ticks set with the *Scale* tab.

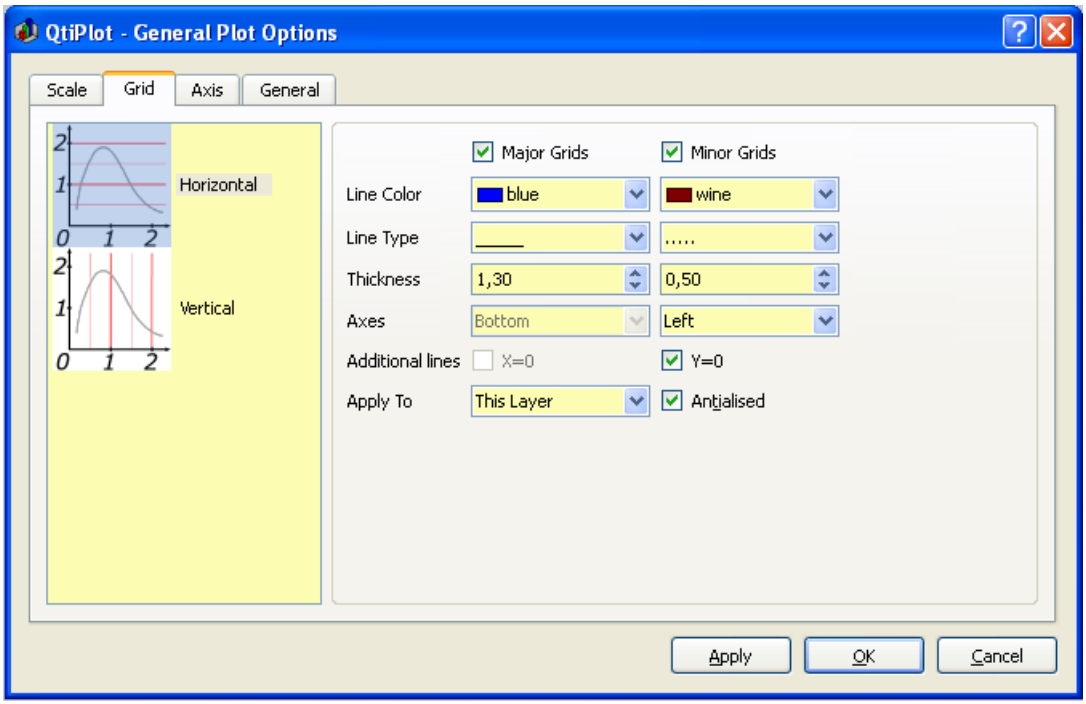


Figure 5.50: General plot options dialog: The Grid Tab.

The third tab is used to modify the setting of the different axes. You must select the axis to be customized in the right window. The label of that axis can then be modified in the title window - see the text options dialog section for more details.

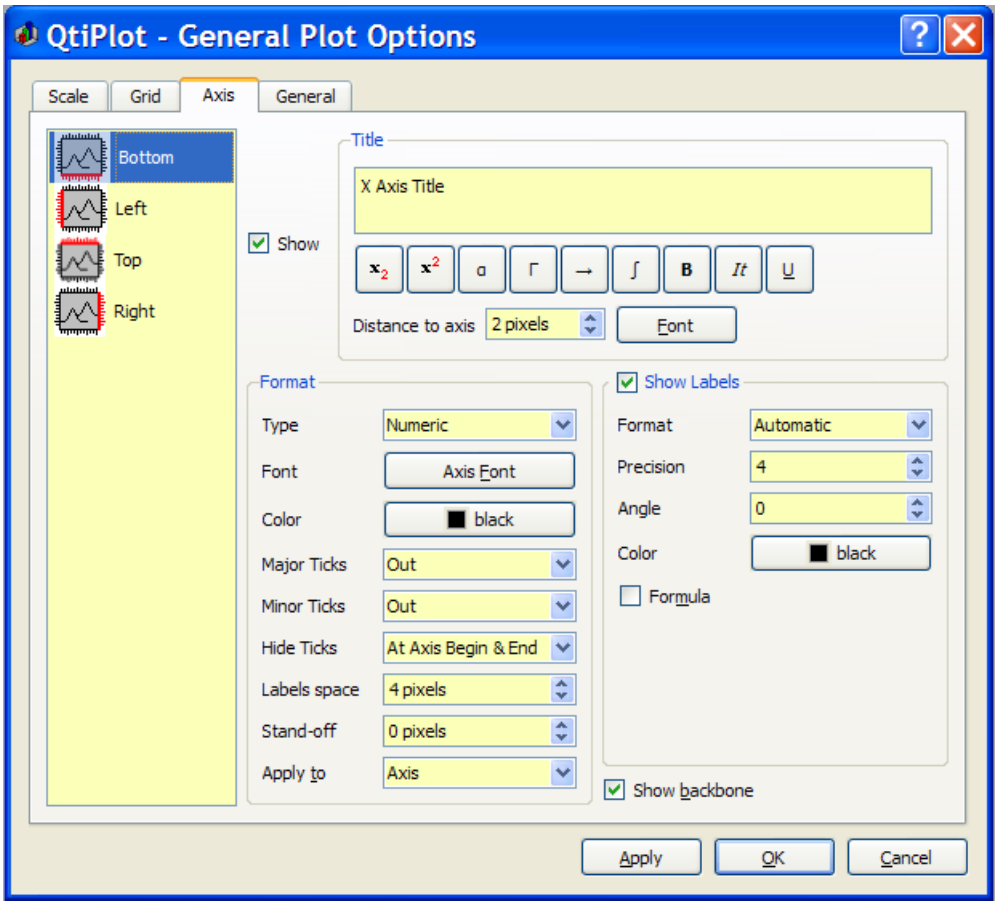


Figure 5.51: General plot options dialog: The Axis Tab.

The *General* settings tab is used to customize global aspects of the plot. The canvas is the area bounded by the axes. You can draw a box around this canvas and define its background color. The background area is the global drawing area of the graph window. You can also define a color border and a background color for this area as well. 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 (manually with the mouse or with the arrange layers dialog.

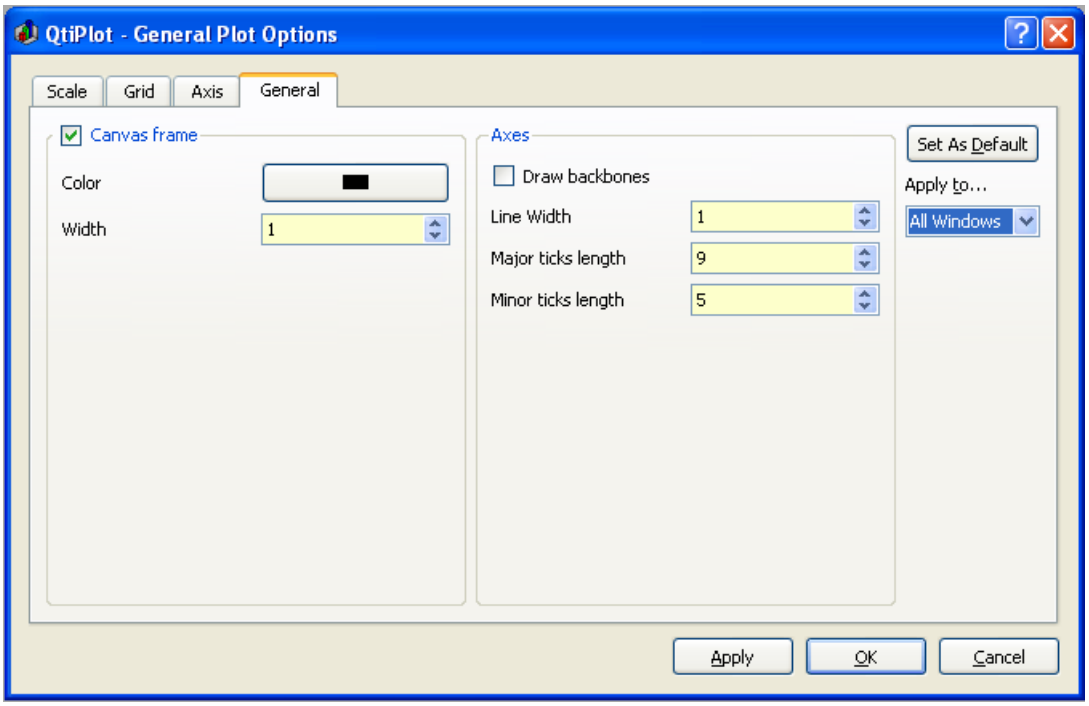


Figure 5.52: General plot options dialog: General Settings.

The parameters in the *Axes* group allow modification of the linestyle used to draw the axes and 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 from the tables available in the current project. First, select the table you want to use from the *Worksheet* combo box and click on *New curve* to create the curve. After that, select at least one column for X and one for Y using the <->X and <->Y buttons. You can also select columns for Z, X-errors and/or Y-errors using the buttons provided for these options. The plot will be created using the default style defined using the '2D Plots -> Curves' tab in the Preferences dialog.

In this example, one curve is selected from the first table, and the other from the second table (with X error bars)

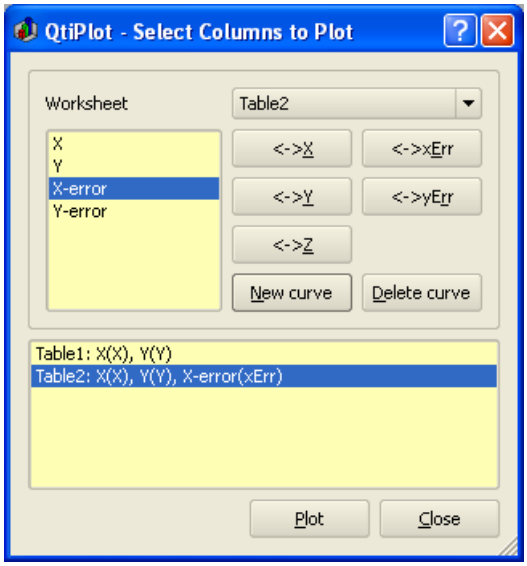


Figure 5.53: The plot wizard dialog box.

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, and to perform basic operations like: renaming, deleting, hiding, resizing, printing, etc. You can also use it to display a list of the dependencies and properties of any element in the project.

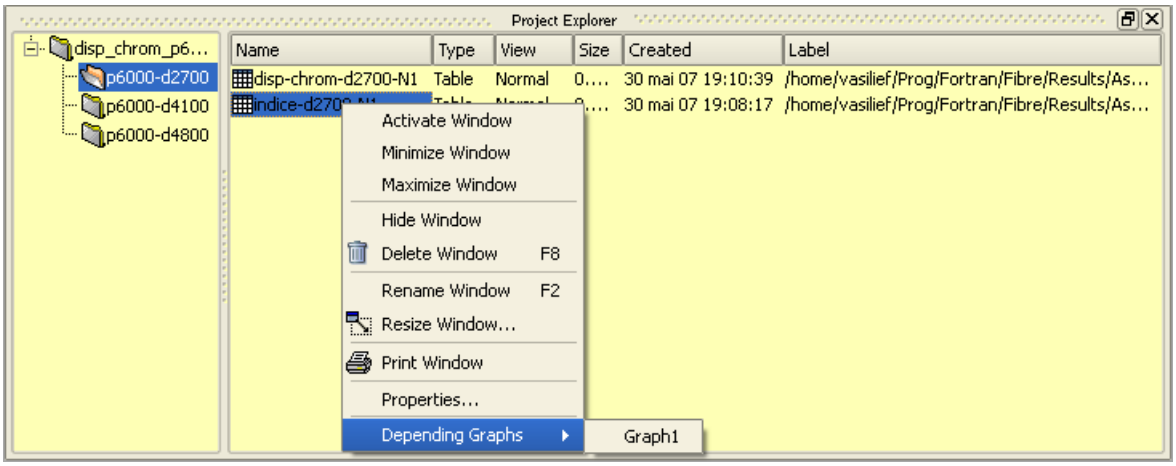


Figure 5.54: The project explorer panel.

5.19 Preferences Dialog

The preference dialog is used to customize the default behavior of QtiPlot. There are six pages of options which are selected using one of icons from the column on the left hand side of the dialog. The text box at the top of the dialog shows the name of the selected page. No matter which page is selected, there are always 4 buttons along the bottom edge of the dialog: *Default Options*, *Apply*, *OK*, and *Cancel*. Changes must be confirmed using either the *Apply* or *OK* buttons. The *OK* button additionally closes

the dialog. Upon confirmation, changes are saved and stored immediately. The *Cancel* button will not undo changes which have already been confirmed but provides a means of exiting the dialog without confirming any changes that are pending. Finally, the *Default Options* button will restore the values of all the items on the page/tab which is currently displayed to the program defaults. This is useful when you've lost track of what you may have changed.

5.19.1 General Preferences

Selecting the  General icon displays the *General* options page. The controls that control the general options are grouped onto a set of 6 tabs. Each tab references a set of related options.

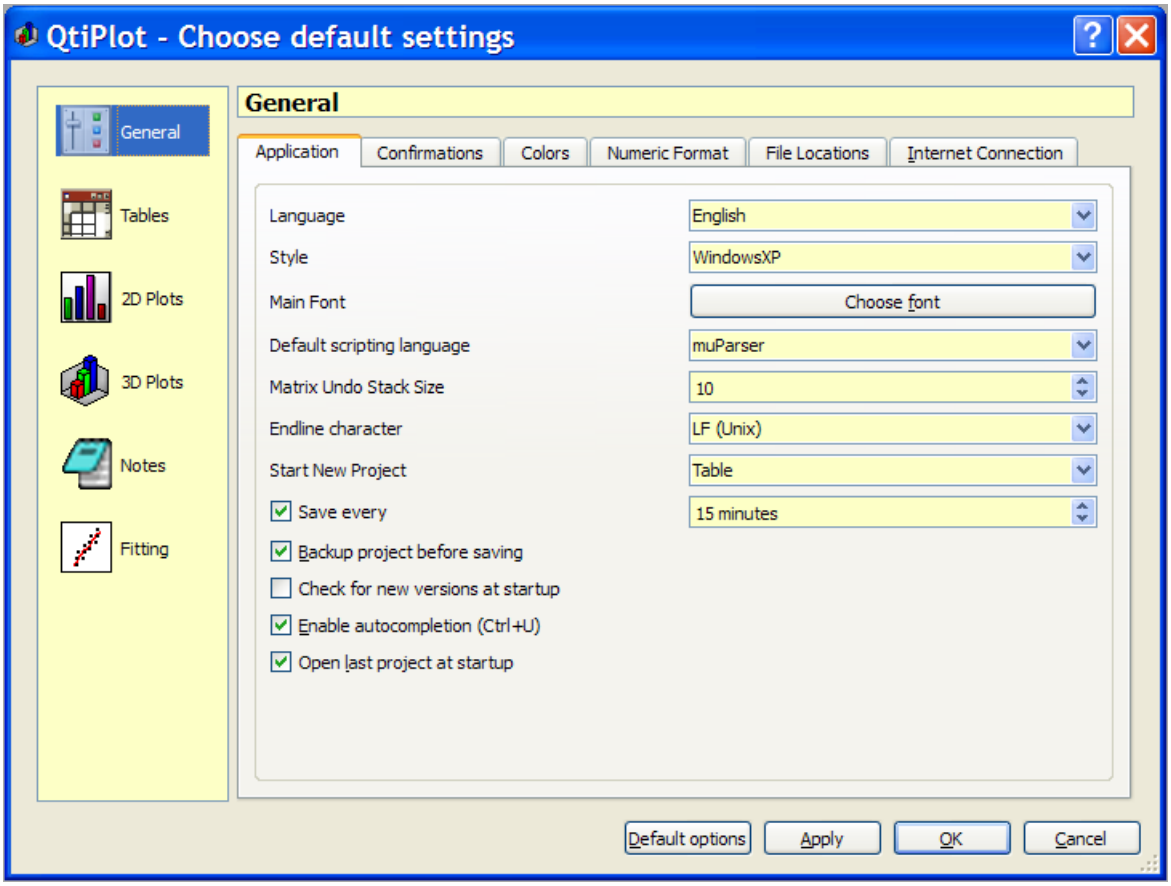


Figure 5.55: The preferences dialog: general parameters for the application.

5.19.1.1 Application Tab

Controls on the *Application* tab are used to set application wide defaults.

The *Language* combo-box lists the translations available in QtiPlot. Select a language from this list. Control names, program labels, etc., will be displayed in this language. All available translations can be downloaded from the following address: http://soft.proindependent.com/translations.html. In order to be loaded by the application, translations must be placed in the folder specified in the File Locations tab. By default, the folder is named *translations* and is located in the same folder as the QtiPlot executable. You can specify a different folder in the *File Locations* tab, if you wish.

The *Style* combo-box defines the style used by QtiPlot for the window decorations. These include stylistic aspects of buttons, dialog boxes, window borders and titles, etc. Available styles are those currently available in the Qt library.

The *Font* chooser selects the font used in the GUI (menus, dialogs, etc). It doesn't apply to plots.

The *Default Scripting Language* combo-box is used to set the scripting language. muParser is the default. Python will also be available if Python support has been compiled into QtiPlot.

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* combo-box 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 set to any one of the following: *Line Feed (LF)*, *Carriage Return + Line Feed (CRLF)* or *Carriage Return (CR)* only.

The *Start New Project* combo-box is used to select what type of child window, if any, is created when a new project is started. The default is for new projects to contain an empty Table window.

The *Save Every* check box is used to turn the auto-save feature on(checked) or off(unchecked). The associated textbox/spin button is used to set the autosave interval. The interval is in minutes. The textbox only accepts positive integer numeric input. All other input is ignored.

The *Backup project before saving* option is used to create a backup copy of the current project before saving a changed project file. This option is enabled when checked.

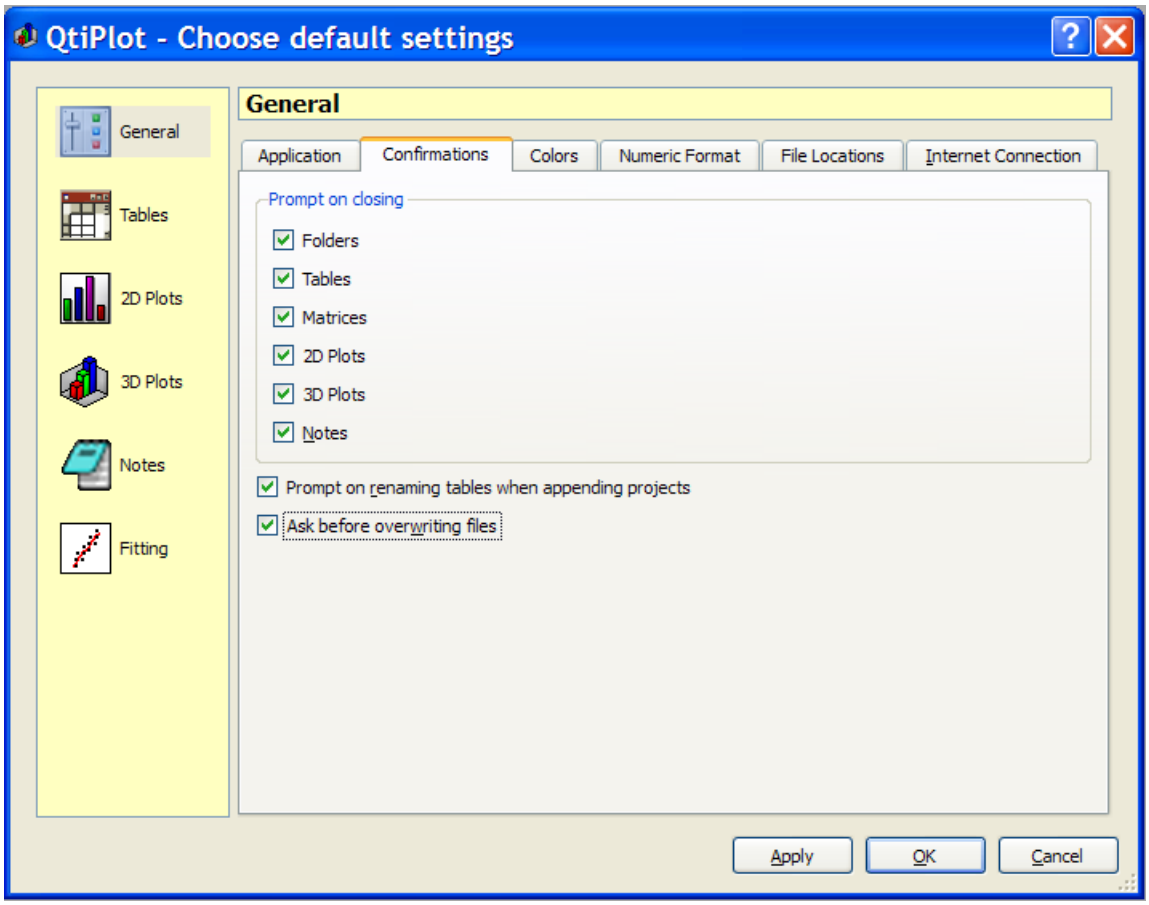
When checked, the *Check for new versions at startup* option will look for program updates on the internet each time the program is started. The default is disabled.

When checked, the *Enable Autocompletion* option enables the autocompletion feature of QtiPlot. Starting with version 0.9.6, autocompletion is enabled by default in all QtiPlot editors (Notes, Script Windows, and 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, below), 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 popped-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.

When checked, the *Open last project at startup* option enables the feature that reloads the last active project when QtiPlot is restarted. The default is enabled.

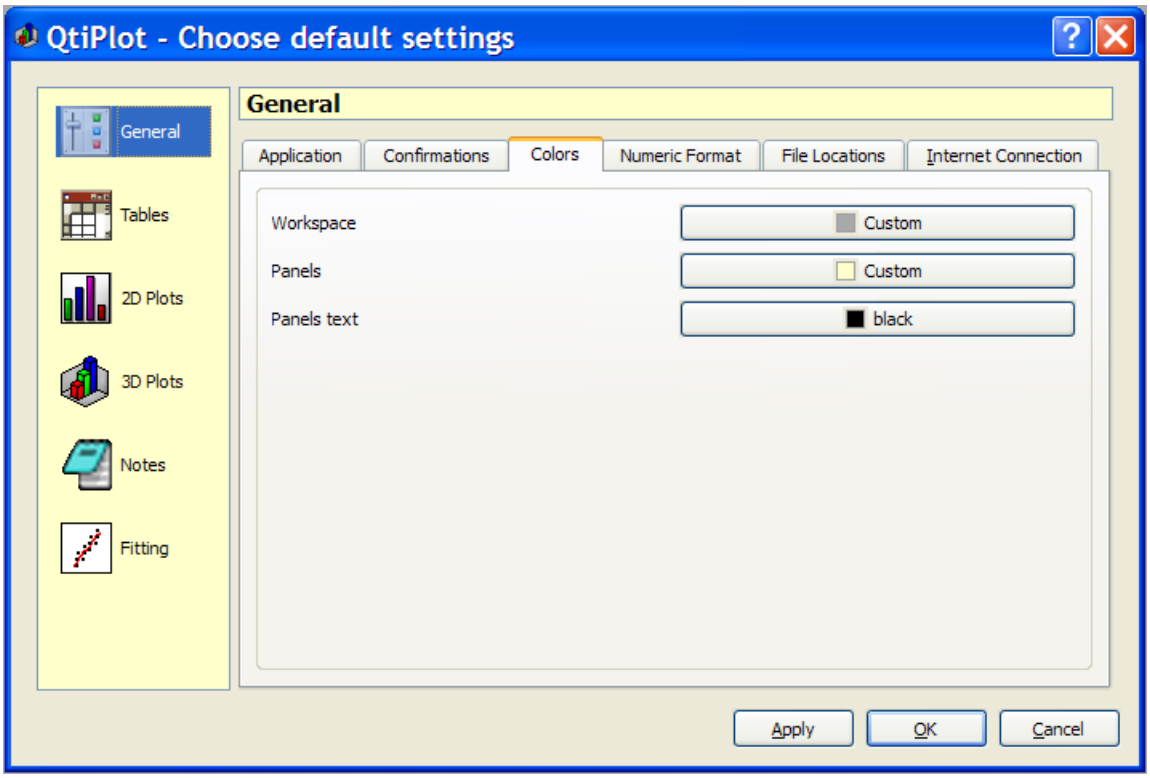
5.19.1.2 Confirmations Tab

The *General Preferences: Confirmations* tab contains a set of 8 check boxes that enable/disable various warning prompts. The first six are warnings given when deleting project windows (Folders, Tables, Matrices, 2D Plots, 3D Plots, and Notes). The remaining 2 are warnings given when 1) renaming or appending windows with names that are already used in the current project, and, 2) when attempting to overwrite an existing file. All warnings are enabled by default.



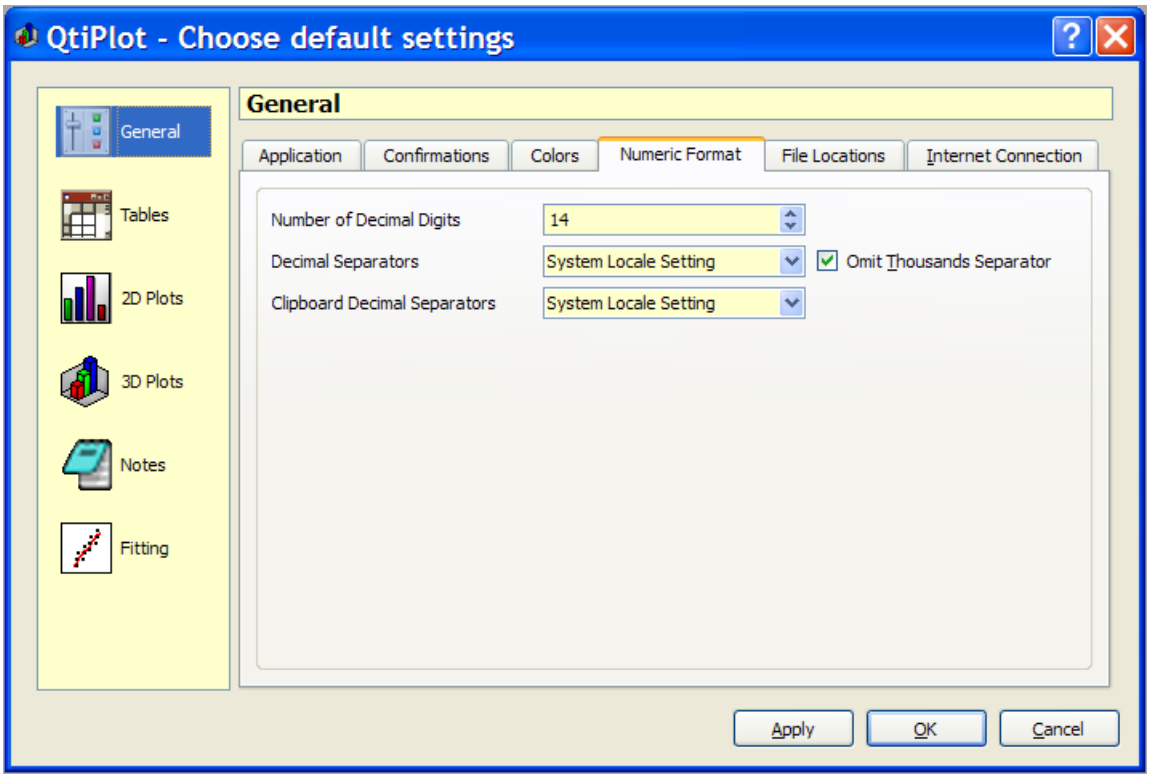
5.19.1.3 Colors Tab

In this tab, you can change the default color for the QtiPlot workspace, the panel background color and the panel text color. Panels refer specifically to the Log Window and the Project explorer.



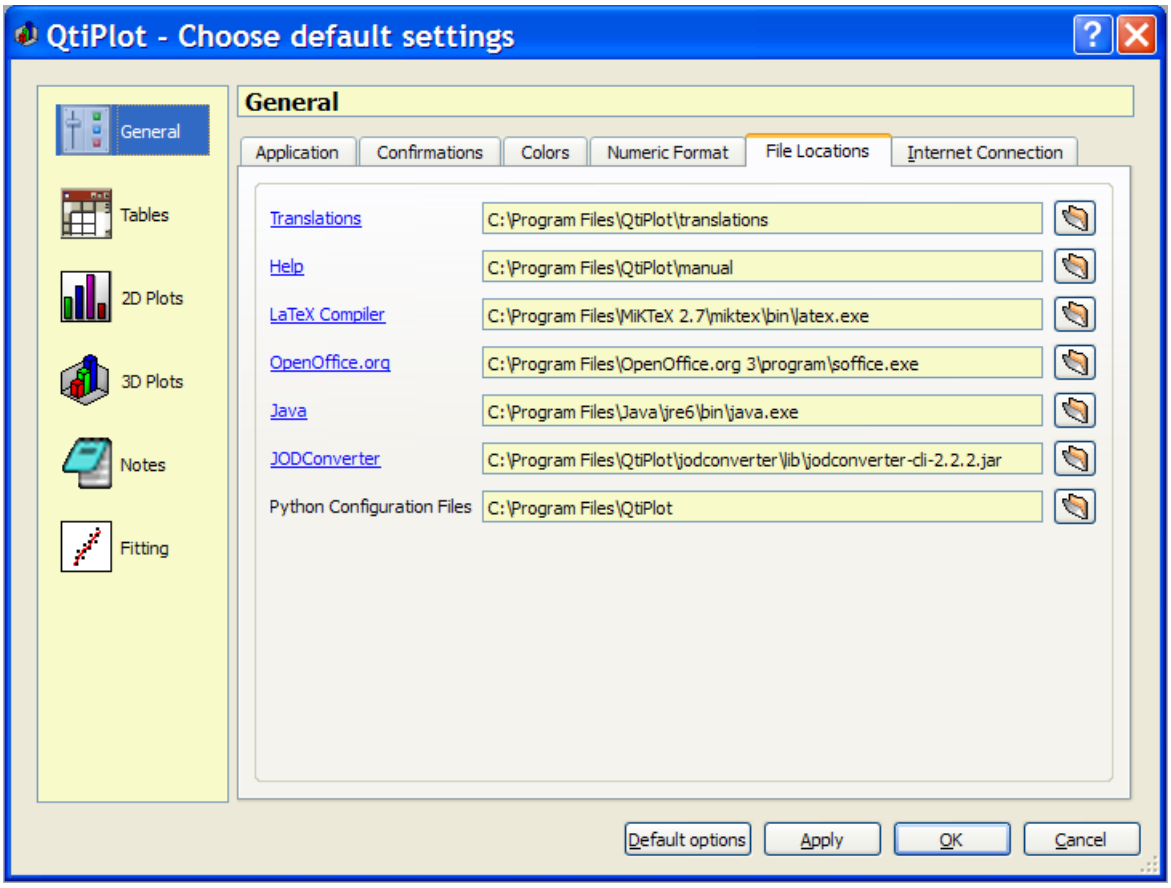
5.19.1.4 Numeric Format Tab

The *Numeric Format* tab allows customizing several aspects of numeric formatting used by QtiPlot. The *Number of Decimal Digits* specifies the default precision used for any calculations applied to your data in Tables and Matrices. The *Decimal Separators* fields allow selection of the characters used as the decimal point and the thousands separator. By default, QtiPlot uses the *locale* settings detected on your system. Separate fields are provided for data in tables/matrices and data copied to the clipboard. The thousands separator can be eliminated completely from tables and matrices by checking the *Omit Thousands Separator* option. QtiPlot will convert all the existing data in your project to the new settings when you click the *Apply* button.



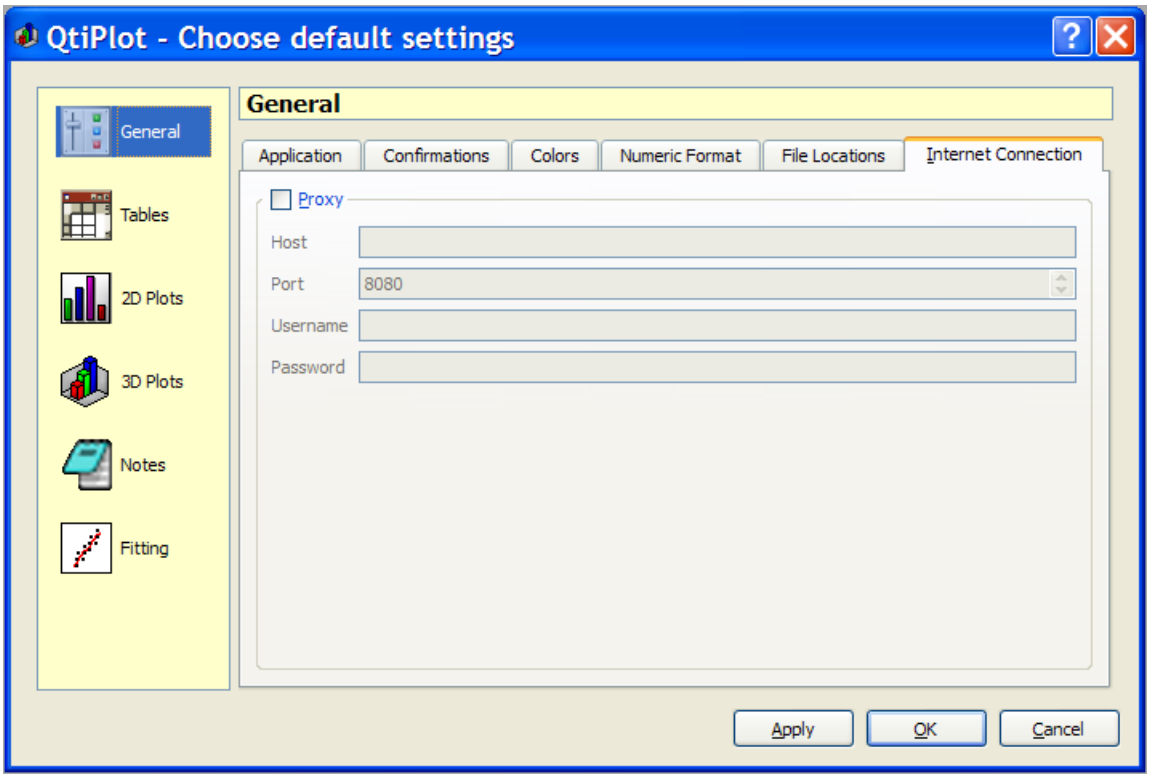
5.19.1.5 File Locations Tab

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*) if QtiPlot was built with Python scripting support. Default folder entries are also provided for the *LaTex Compiler* and for *scripts* that are to be loaded at program startup. The *LaTex Compiler* folder only has meaning if QtiPlot was compiled with *LaTex* support.



5.19.1.6 Internet Connections Tab

Settings on the *Internet Connection* tab are only needed if you connect to the internet via a proxy server. If you don't now how to set these options, contact your Network Administrator or other suitably knowledgeable person.



5.19.2 Tables Preferences

Selecting the  Tables icon opens the second page of the preferences dialog which allows customizing default aspects of tables: background, text colors, and fonts for tables and labels. By checking the *Display Comments in Header* option, column comments will also be displayed in the table header, below 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.

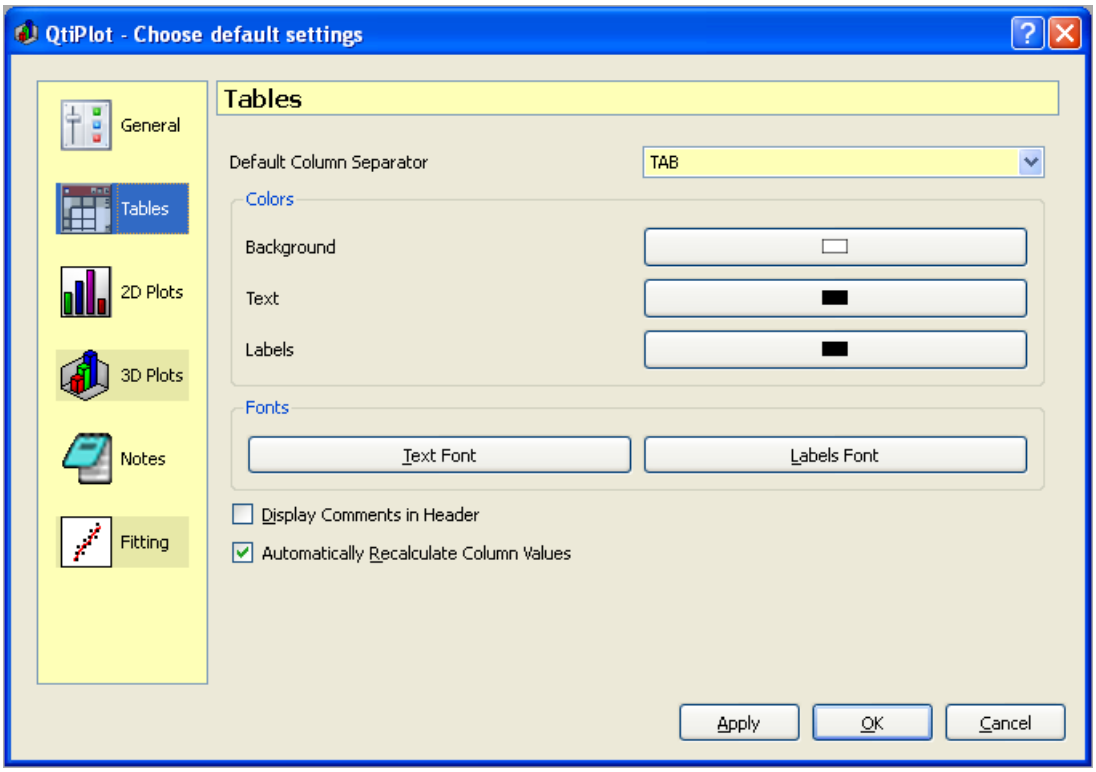


Figure 5.56: The preferences dialog: table options.

5.19.3 2D Plot Preferences

Selecting the  2D Plots Icon opens the third page of the preferences dialog. This set of options is used to customize default aspects of *2D plots*.

5.19.3.1 Options Tab

The *Options* tab is used to modify some general options. Most of the changes made to these options will be applied only to newly created plots. Changing a few of these options, such as plot axis *Autoscaling*, *Antialiasing* of curves and the behavior on resize events will affect extant plots.

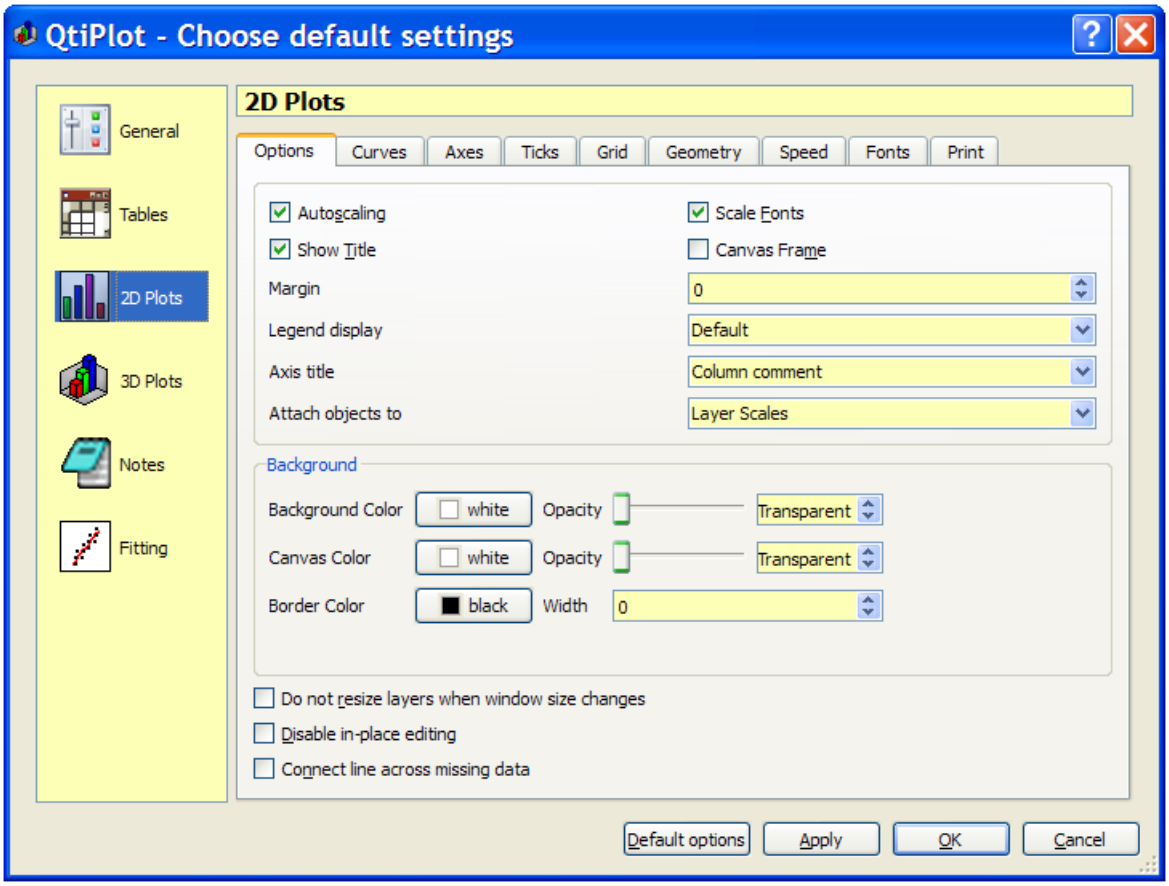
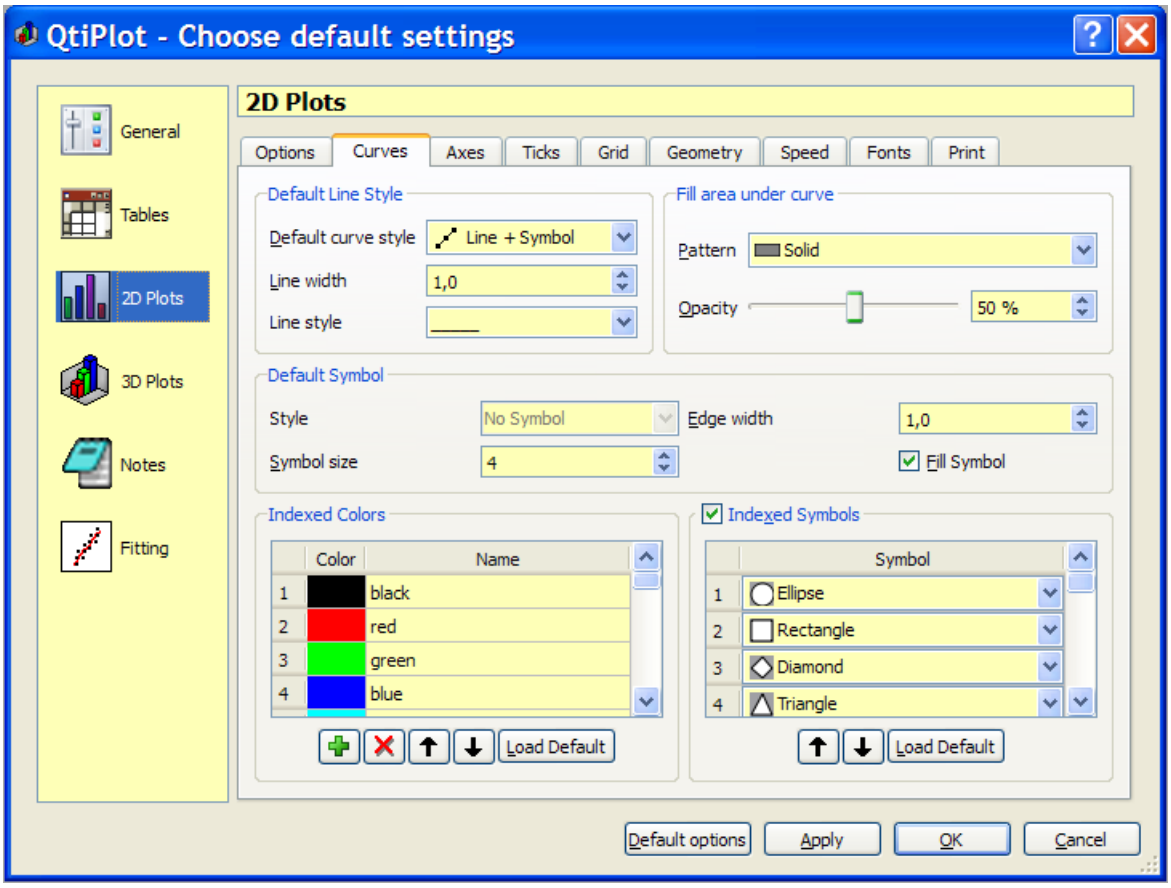


Figure 5.57: The preferences dialog: 2D plot options.

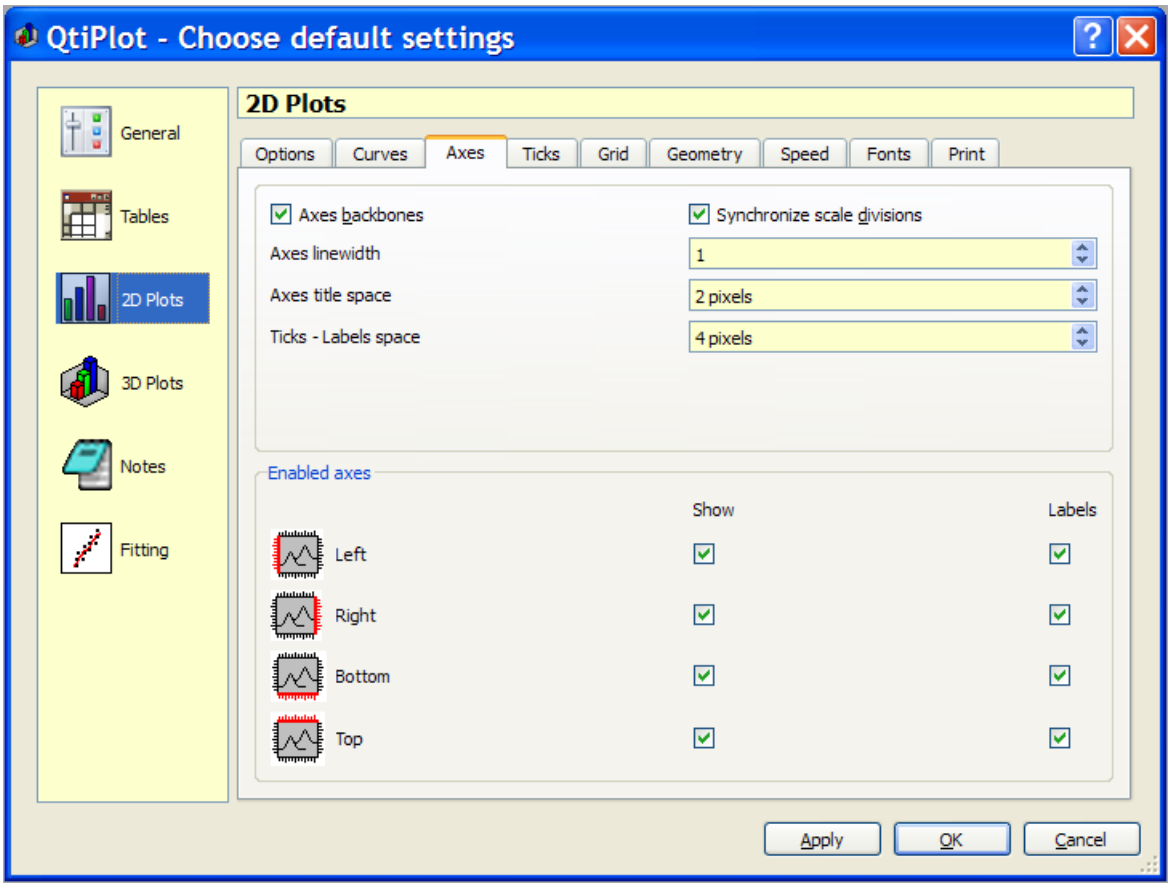
5.19.3.2 Curves Tab

The *Curves* tab contains a large number of controls that define the default style used when creating a new plot. The operation of these controls should be self evident.



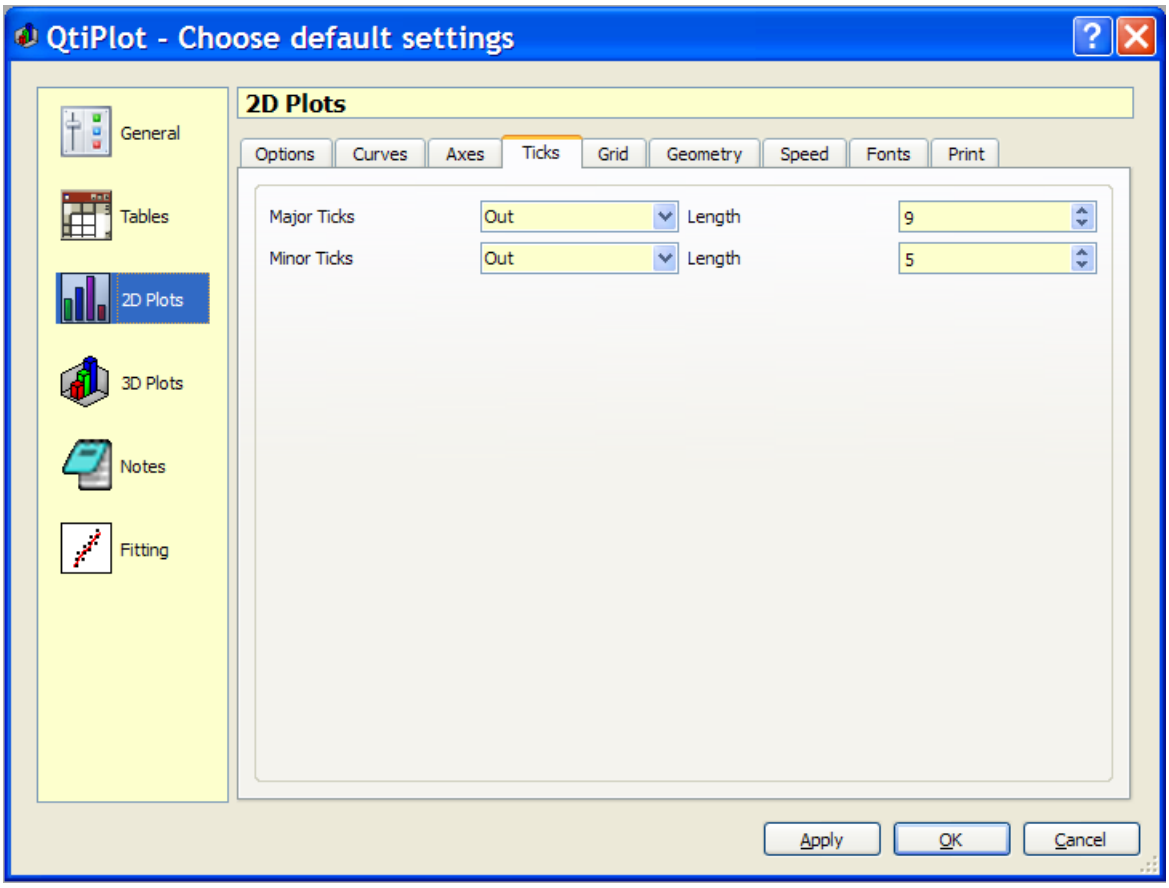
5.19.3.3 Axes Tab

The *Axes* tab allows specification of which plot axes will be displayed in a new layer and the dominant stylistic aspects of the axes.



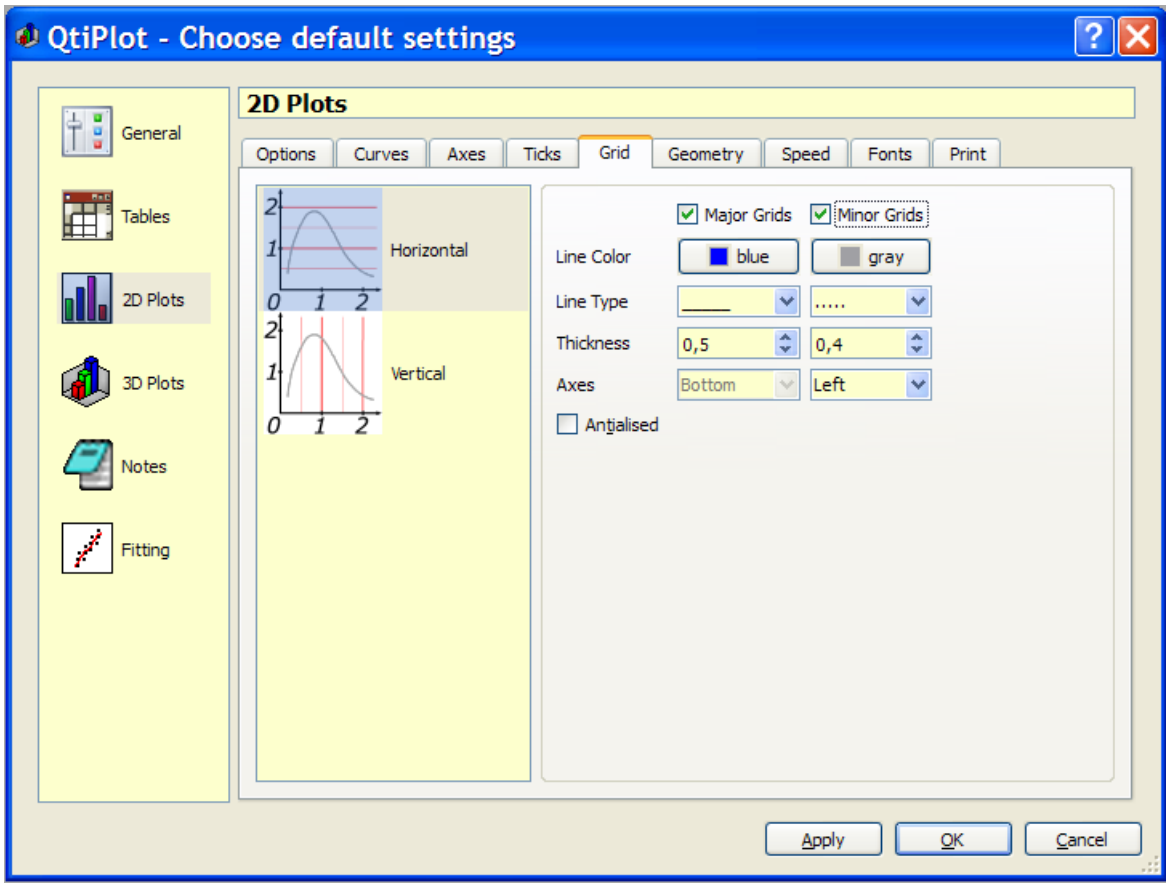
5.19.3.4 Ticks Tab

The *Ticks* tab defines the default style for axis ticks drawn on newly created plots.



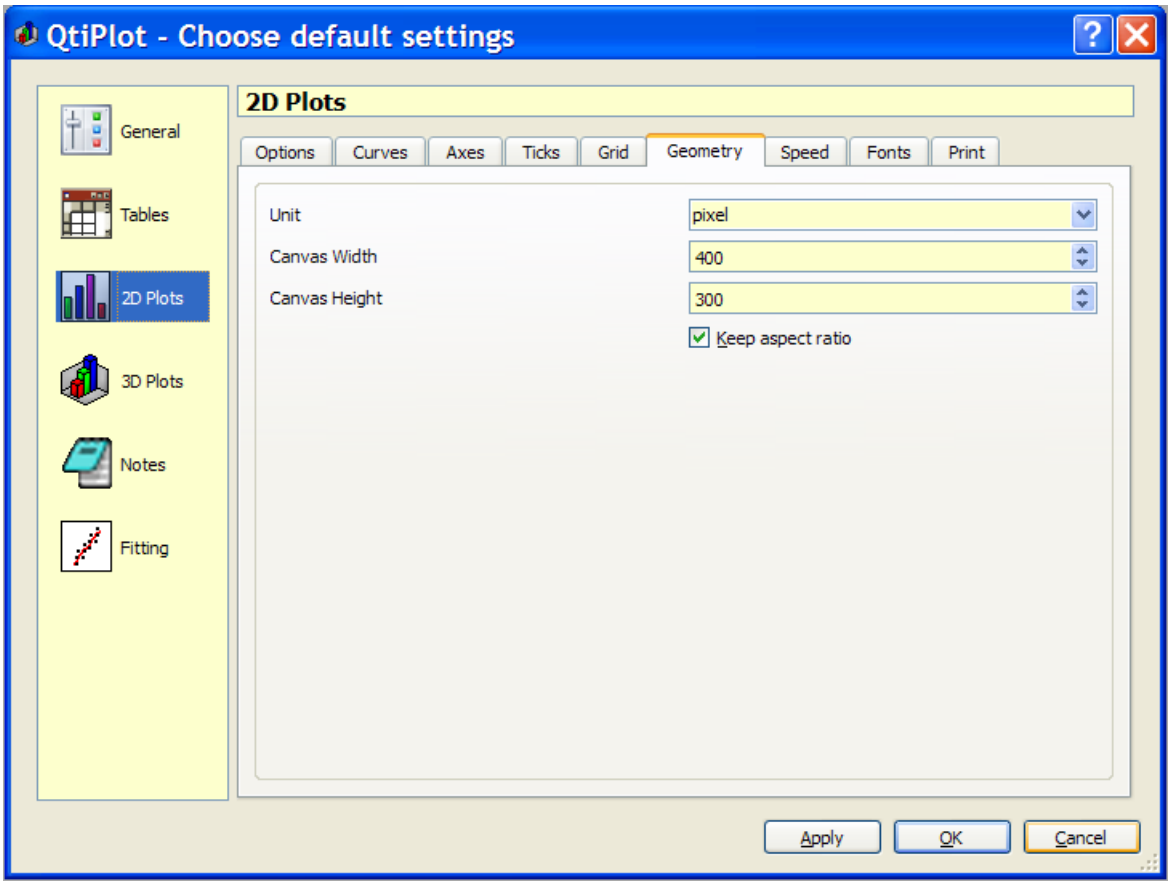
5.19.3.5 Grid Tab

The *Grid* tab defines the default aspect of plot layer grids.



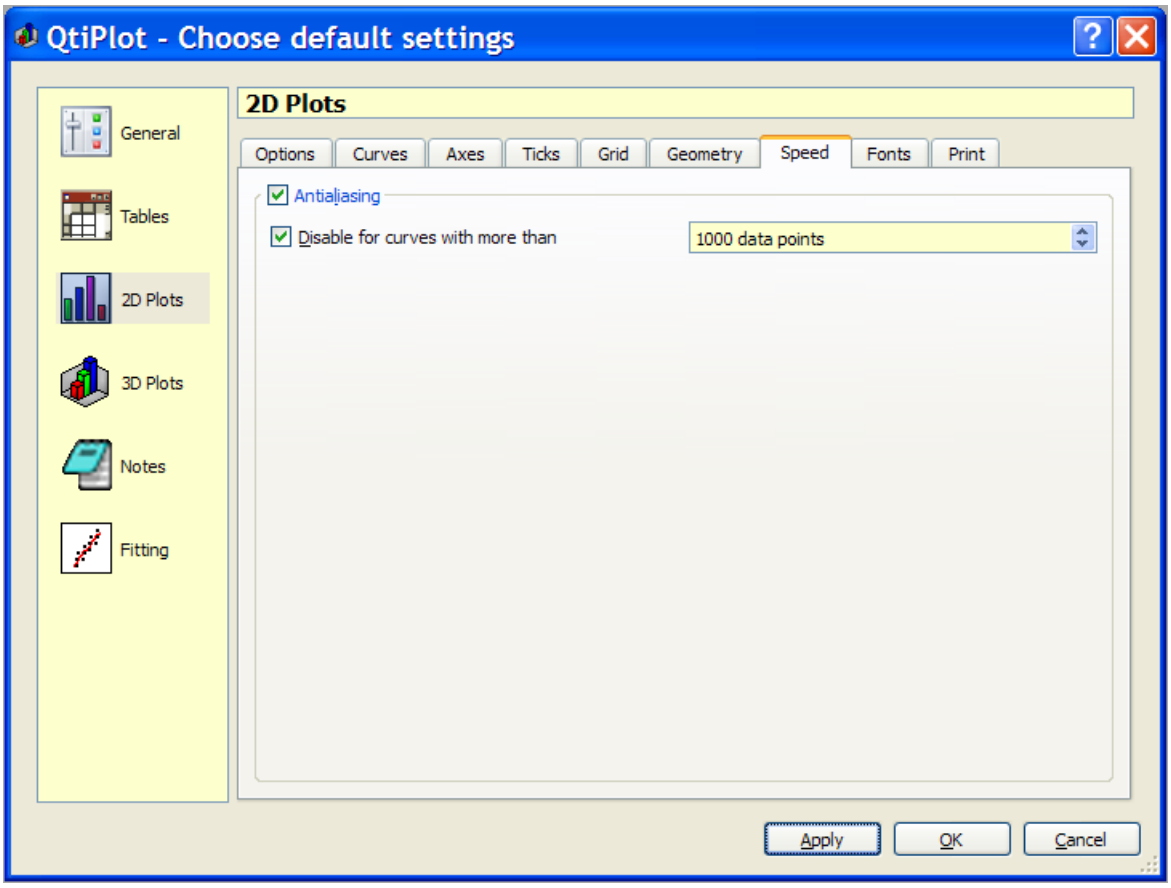
5.19.3.6 Geometry Tab

The *Geometry* tab defines the default size for the drawing area of a plot layer When the *Keep aspect ratio* option is checked, changing either the width or height will change proportionally change the other.



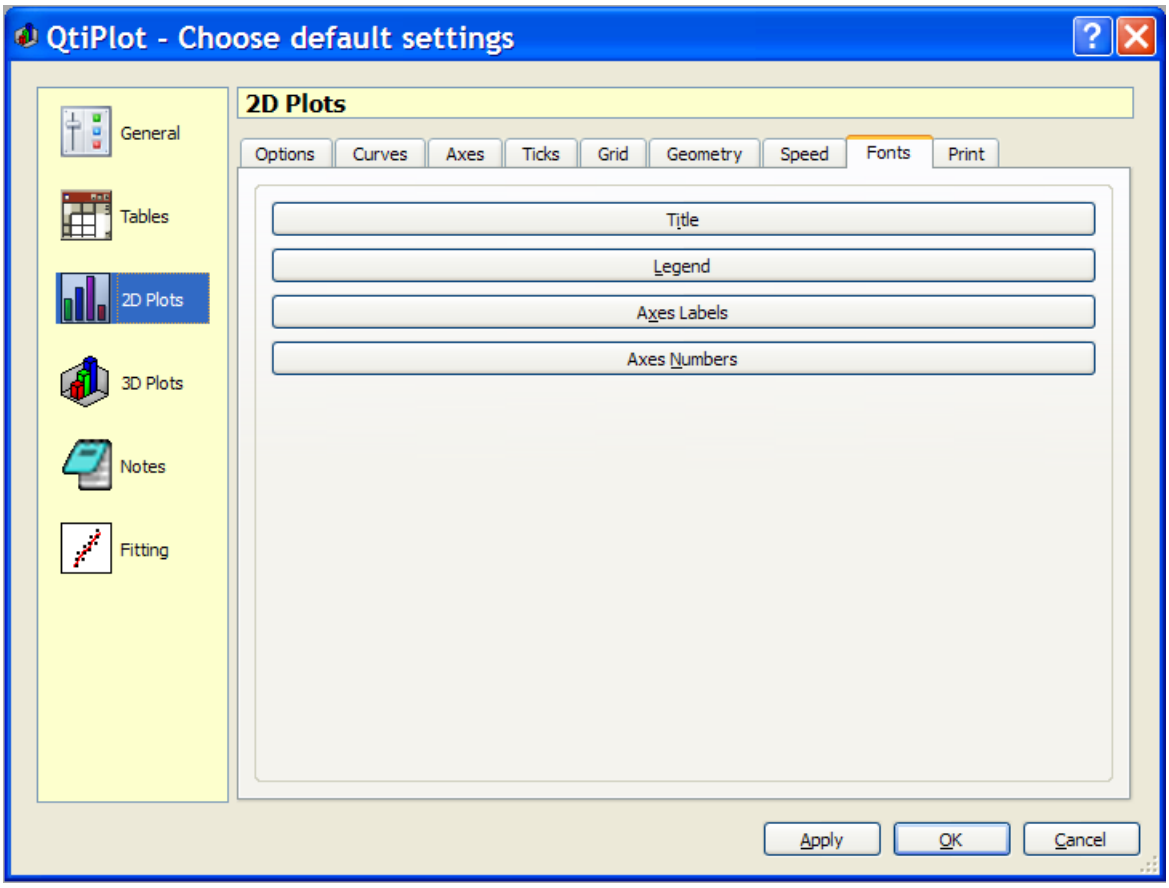
5.19.3.7 Speed Tab

The *Speed* tab lets the user enable/disable antialiasing when drawing/redrawing 2D plots. Antialiasing is a major source of slow-down when rendering 2D plots. Unchecking the *Antialiasing* checkbox disables antialiasing for all curves, which probably will only be needed in extreme circumstances. Checking the *Disable for curves with more than* checkbox will disable antialiasing only for curves having data sets larger than the threshold set with the textbox to the right of the *Disable for curves with more than* checkbox. Disabling this option is probably not a good idea. The default is for both of these options to be enabled, with a threshold of 1000 data points. Proper setting of these options is essential to keeping the application responsive.



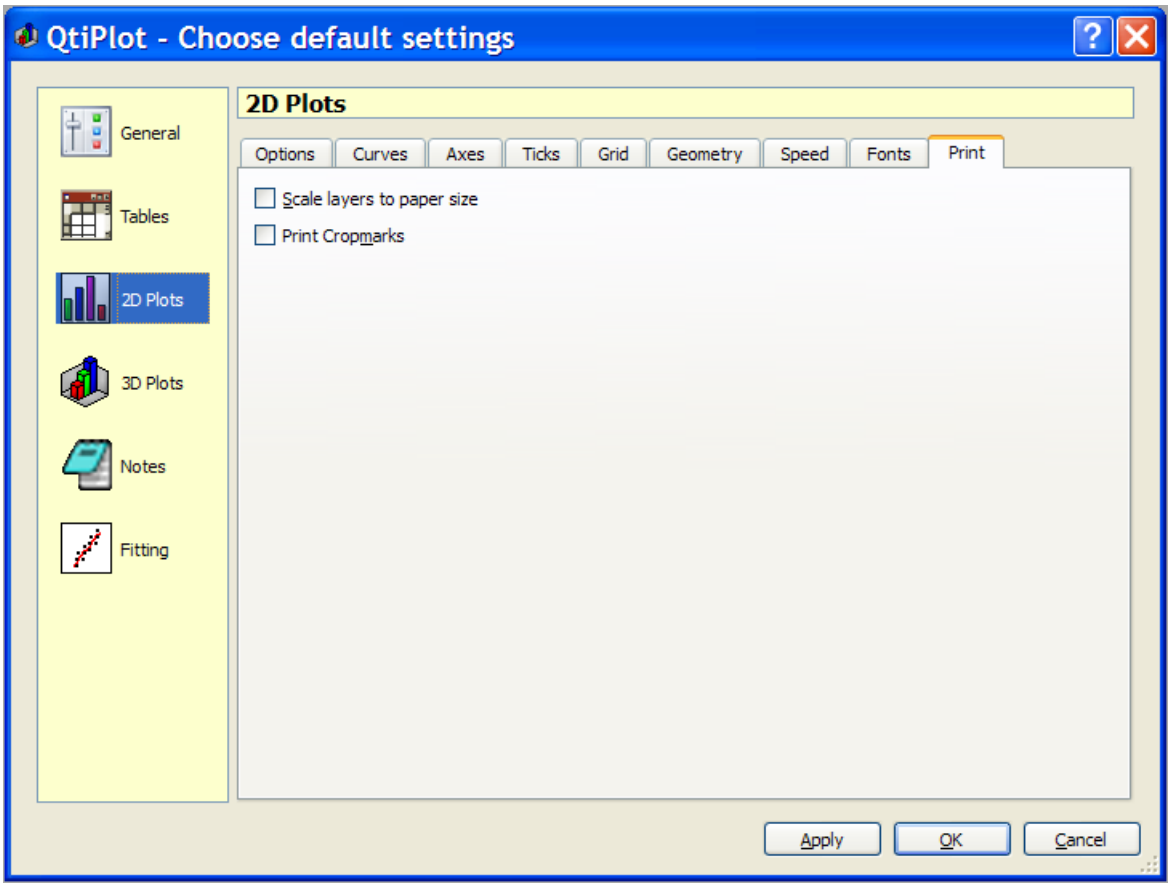
5.19.3.8 Fonts Tab

The *Fonts* tab defines the default fonts used when creating new plots. Options are provided for the plot *Title*, plot *Legend*, *Axes Labels*, and *Axes Numbers*.



5.19.3.9 Print Tab

The *Print* tab allows you to control a few default options that are used when printing 2D plots. If you want layers to be printed with their original dimensions, you must be sure to uncheck the *Scale layers to paper size* option. Checking the *Print Cropmarks* option ensures that some visible marks will be drawn around the borders of the plot.



5.19.4 3D Plot Preferences

Selecting the  3D Plots Icon opens the fourth page of the preferences dialog. This set of options is used to customize default aspects of three dimensional plots. Most of the options are self-explanatory. However, the *Resolution* option needs clarification. This option is more or less akin to a speed drawing mode, which can be very useful when working with large data sets. Larger values of the *Resolution* option result in fewer data points being drawn on 3D plots, and therefore a higher drawing speed. When *Resolution* is set to 1, all data points are drawn.

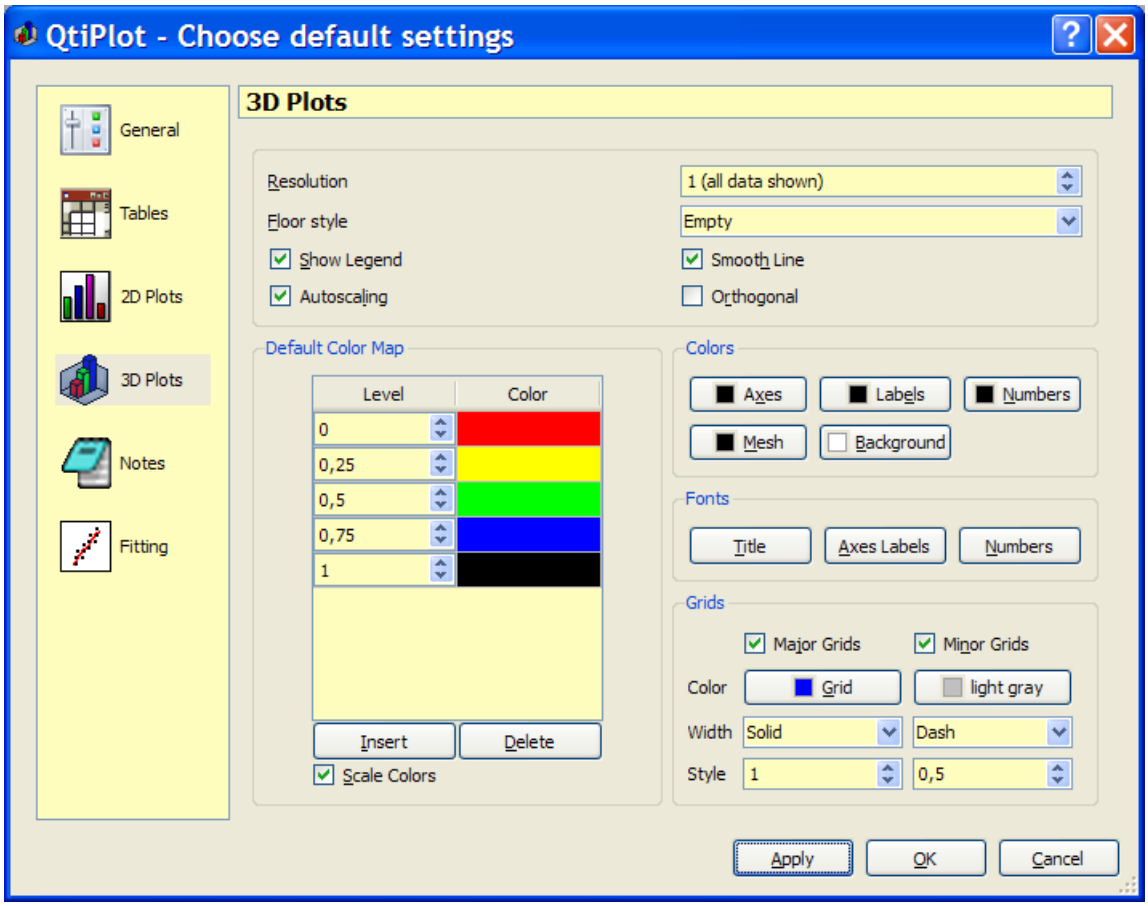


Figure 5.58: The preferences dialog: 3D plot options.

5.19.5 Notes Preferences

Selecting the Notes Icon opens the fifth page of the preferences dialog. This set of options is used to customize some of the default characteristics of the text editors, such as the length of the *TAB* character and the font. The user can also specify whether or not line numbers should be displayed. Displaying the line numbers can be particularly helpful when debugging Python scripts.

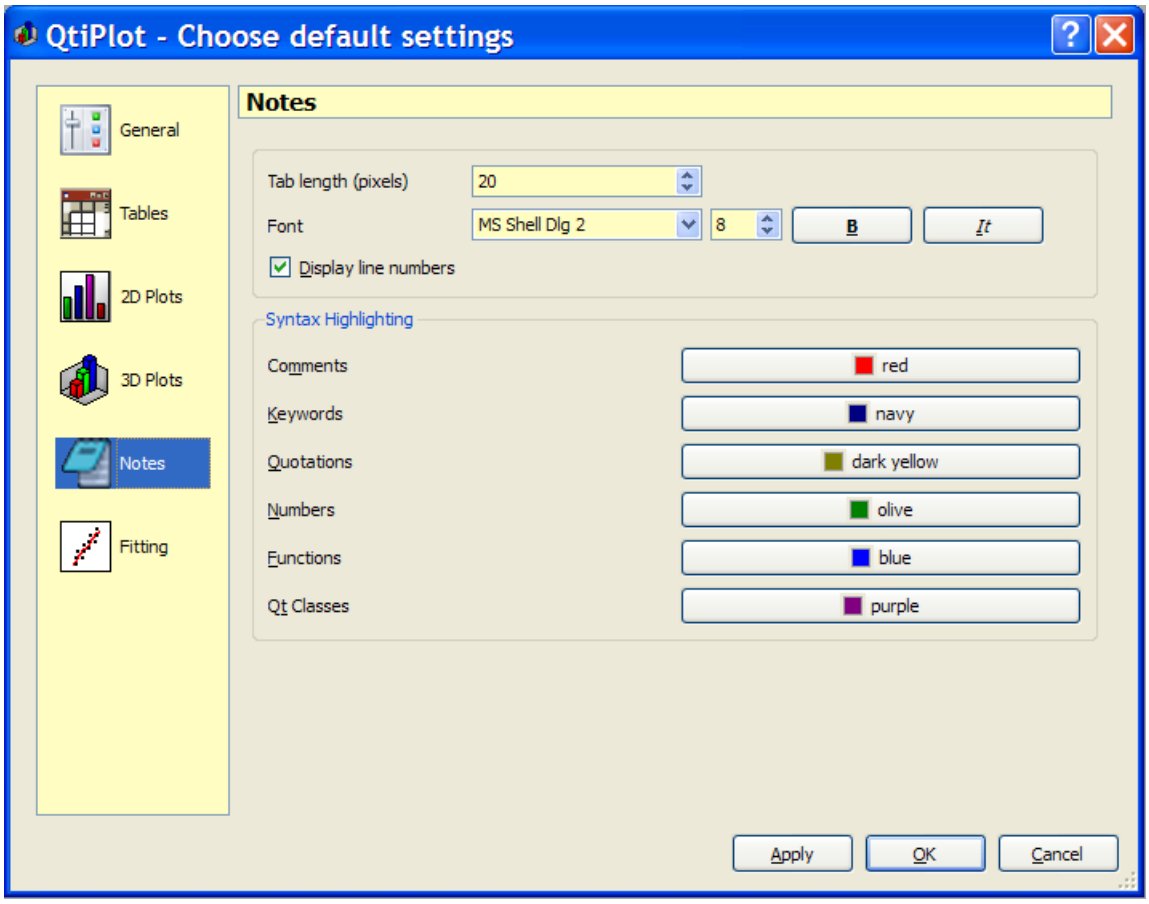


Figure 5.59: The preferences dialog: note options.

5.19.6 Fitting Preferences

Selecting the  Fitting Icon opens the sixth page of the preferences dialog. This page is used to set default fitting options. Most of the options are standard and straightforward. The *Generated Fit Curve* options may be confusing at first glance. While it may be typical to plot a fit curve as y=f(x) using the original X data that was used in the fitting operation, QtiPlot provides the alternative (by selecting the *Uniform X Function* option) of plotting the curve using a user specified number of X data points (default=100) uniformly spaced over the X range of the fit. Since linear fits are completely defined by 2 points, you can also have QtiPlot default to simply plotting linear fits using 2 data points by checking the *2 points for linear fits* option.

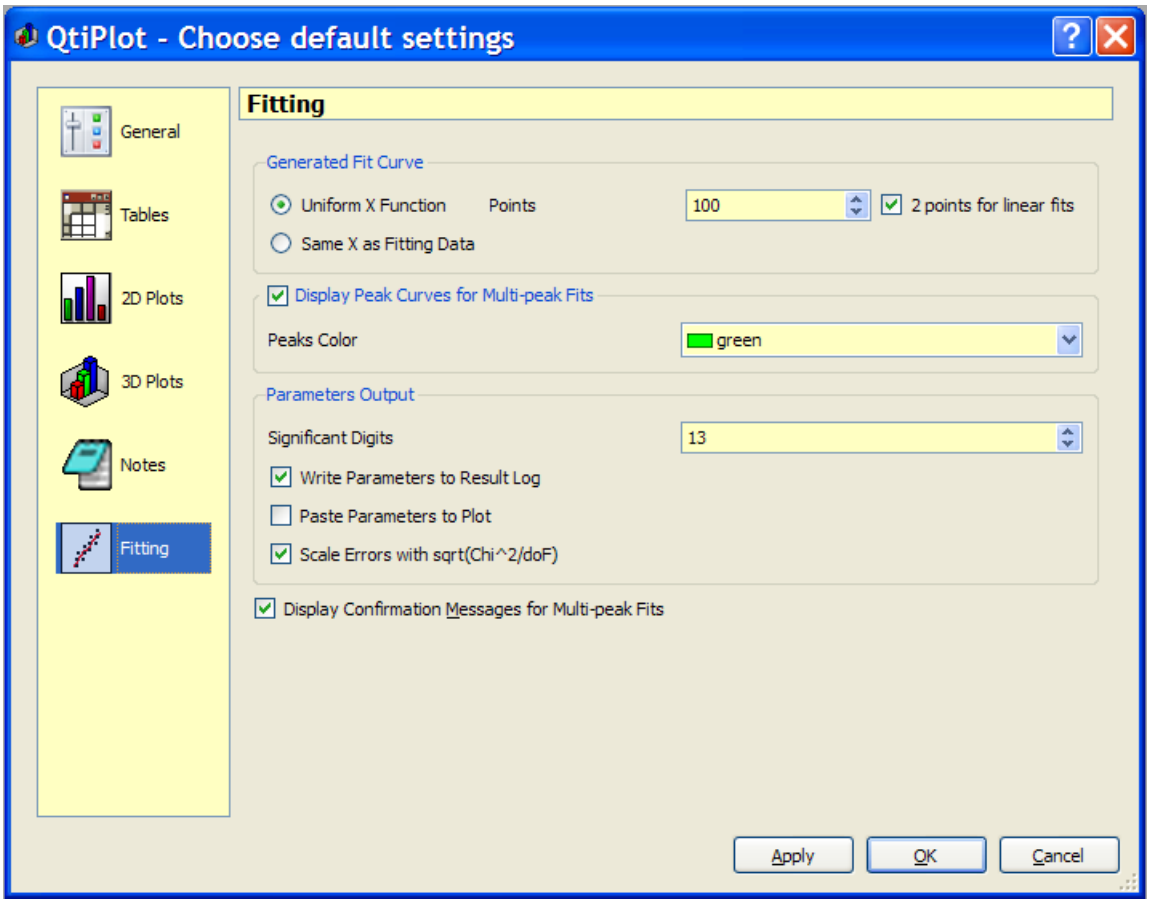


Figure 5.60: The preferences dialog: fitting options.

5.20 Printer-setup

This dialog box is opened by the Print command from the File menu or by entering the Ctrl-P key. It is used to print the selected window (plot or table) and its exact appearance will depend both upon your operating system and the printers that you have available. The following screenshot shows the dialog on a Linux system with one printer that uses the KDE window manager.

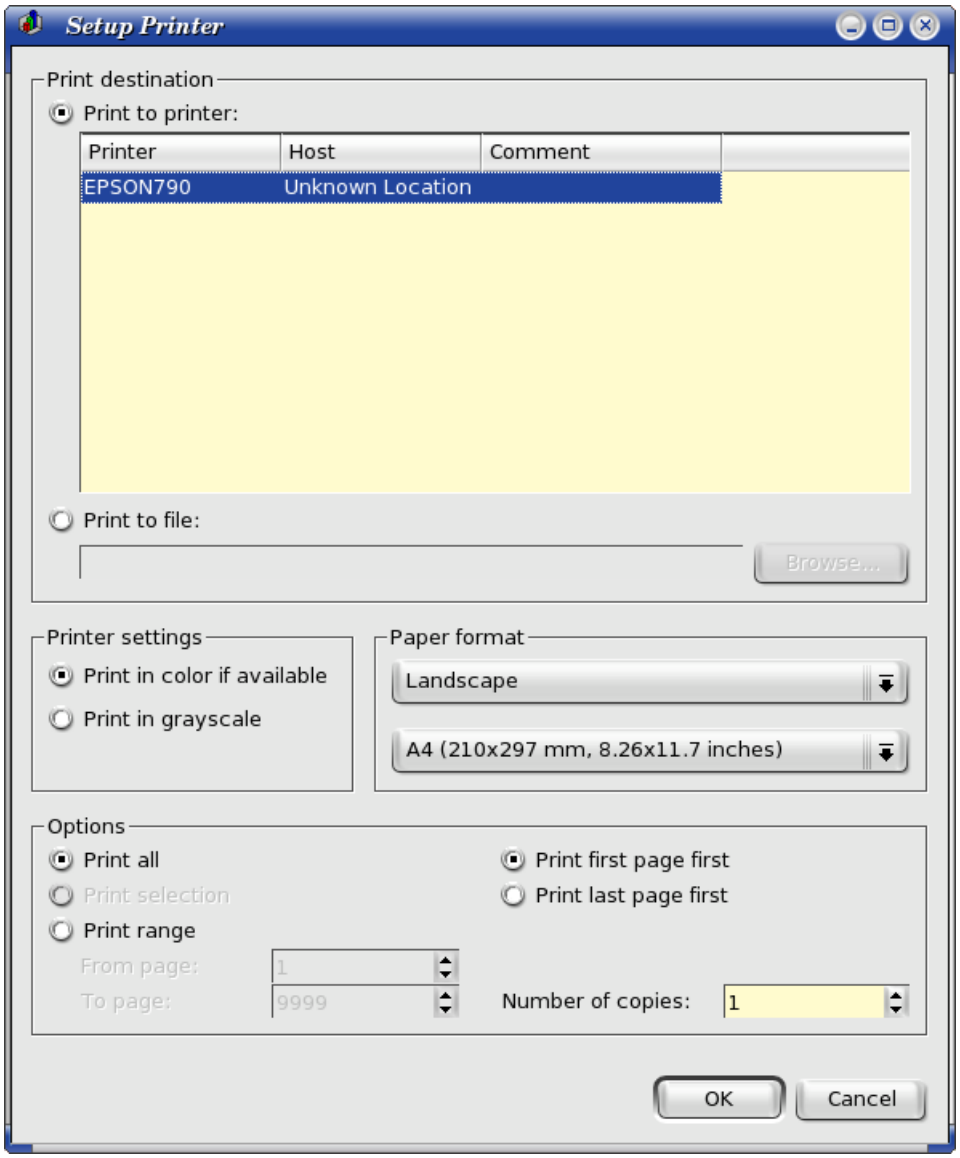


Figure 5.61: The **Print** dialog.

5.21 Set Column Values

This dialog is activated with the Set Column Values... command from the Table menu or by entering the shortcut Alt-Q. It provides for filling a column with the result of a function evaluation.

The available mathematical functions (assuming you are using the default scripting language, muParser) are listed in the *function selector* combo-box next to the *Add function* button. Details of these functions and supported mathematical operators are listed in the muParser section in the chapter on *Mathematical Expressions and Scripting*. The special function, *col(c)*, is used to access the values of column `c`, where `c` is the column's number (as in *col(2)*) or its name in double quotes (as in *col("time")*). You can also obtain values from other tables using the *tablecol(t,c)* function, where `t` is the table's name in double quotes and `c` is the column's number or its name in double quotes (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 starting and ending row, respectively.

Using Python as scripting language gives you even more possibilities. Not only can you use arbitrary Python code in the function body, but also access other objects within your project. For details, see the section on Pythonin the chapter on *Mathematical*

Expressions and Scripting. Nevertheless, even though Python allows for a more powerful syntax, it can be quite slow for very large tables. Therefore you might want to use muParser instead, even when Python is set as the default script engine for your current project, Checking the *Use built-in muParser* option will force the use of muParser as the scripting engine. This greatly increases the speed of the evaluation for single line expressions.

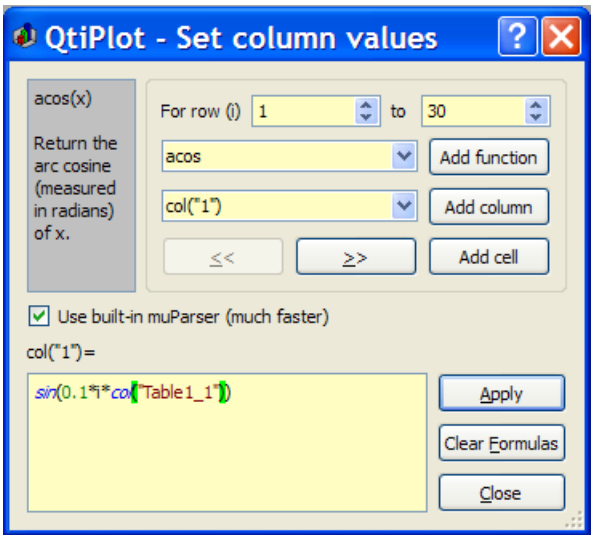


Figure 5.62: The **Set Column Values...** dialog.

Warning: When you make changes to the values in a table which contains "filled" values, the dependent values are not recomputed unless the *Automatically Recalculate Column Values* option is checked in the Tables tab of the *Preferences* dialog. If this option is unchecked, you will have to explicitly tell QtiPlot to recalculate individual cells or whole columns or rows by selecting "Recalculate" from their context menu or by pressing Ctrl-Return.

5.22 Set Matrix Dimensions

This command appears in the Matrix menu. It allows direct specification of 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 new plots as the initial scale ranges. They are also available via the *x* and *y* variables if you choose to define the contents of the matrix with the Set Values Dialog.

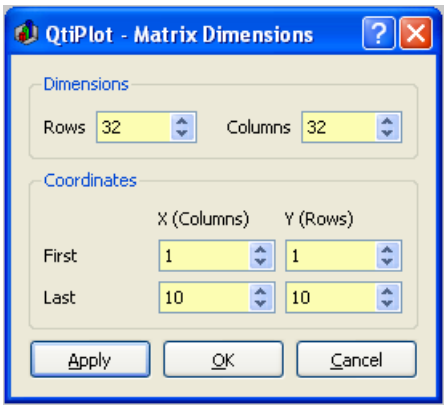


Figure 5.63: The **Set Dimensions...** dialog for matrix.

5.23 Import ASCII files

This dialog is activated by selecting the Import -> Import ASCII... command 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.

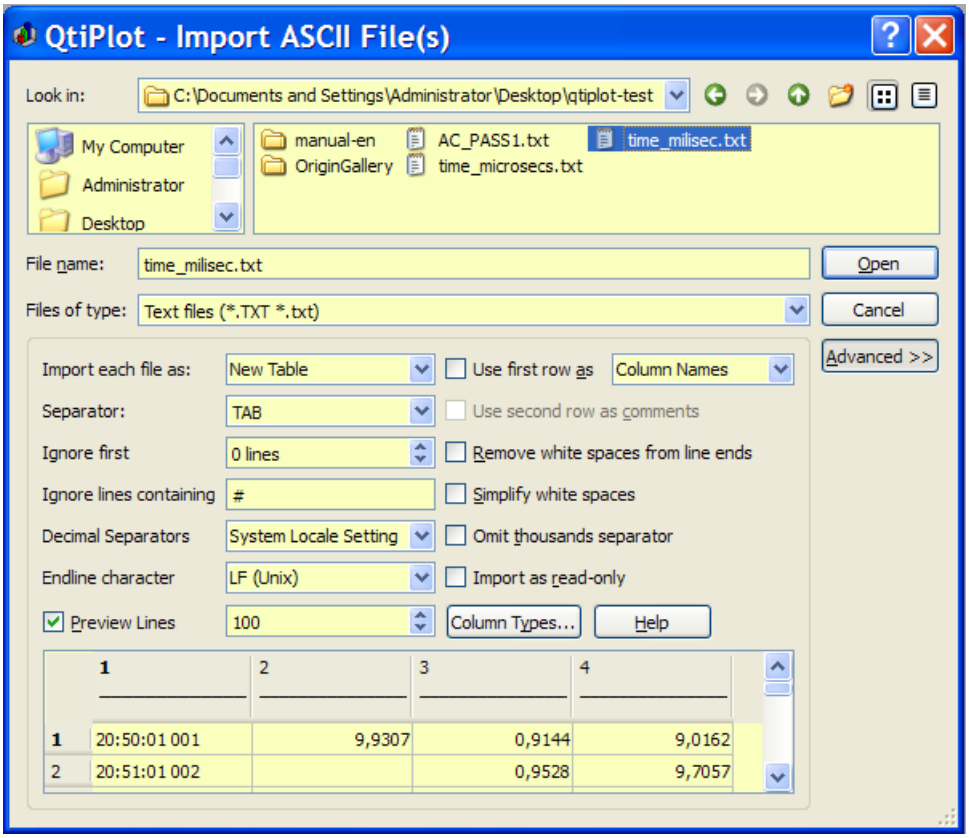


Figure 5.64: The dialog box.

You can define a custom file filter in the *Files of type:* field in order to preselect files. For example, typing *.mydata and then the "Enter" key will set a filter that will show only files of the form "XXX.mydata" in the directory view.

Concerning the character which is used to separate column values, 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 begin with several lines describing your experiment, you may skip the first n lines of the file. You can also skip all lines starting with a comment character/string. The comment string can be customized using the *Ignore lines starting with* text edit box.

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

If the files you want to import were created using a different convention for the decimal and thousands separator characters than the current setting, then the *Decimal Separators* option can be used to configure QtiPlot to use a matching convention. For example, when a comma character is used for the decimal point instead of the dot character, simply change the *Decimal Separators* option to match. You can have QtiPlot ignore the Thousands separator altogether 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 date/time columns.

5.24 Matrix Properties

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

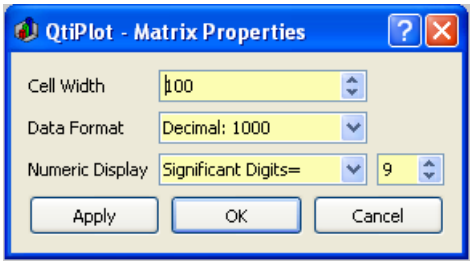


Figure 5.65: The **Set Properties...** dialog for matrices.

5.25 Set Matrix Values

This command is located in the Matrix menu. It allows filling in a matrix with the results of evaluating 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.

Functions can span several lines. The available intrinsic functions are listed in the muParser section of the Mathematical Expressions and Scripting chapter.

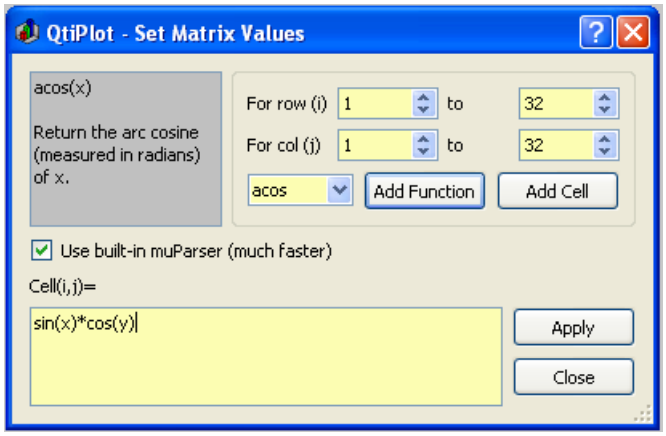


Figure 5.66: The **Set Values...** dialog for matrix.

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

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

in preference for something like the following:

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

which will greatly increase the speed of evaluation!

5.26 Surface plot options

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

5.26.1 Scale Tab

The first tab is used to modify the X, Y, and Z ranges. It also allows specification of the number of labels (major ticks) to be drawn on the axes, as well as the number of minor ticks.

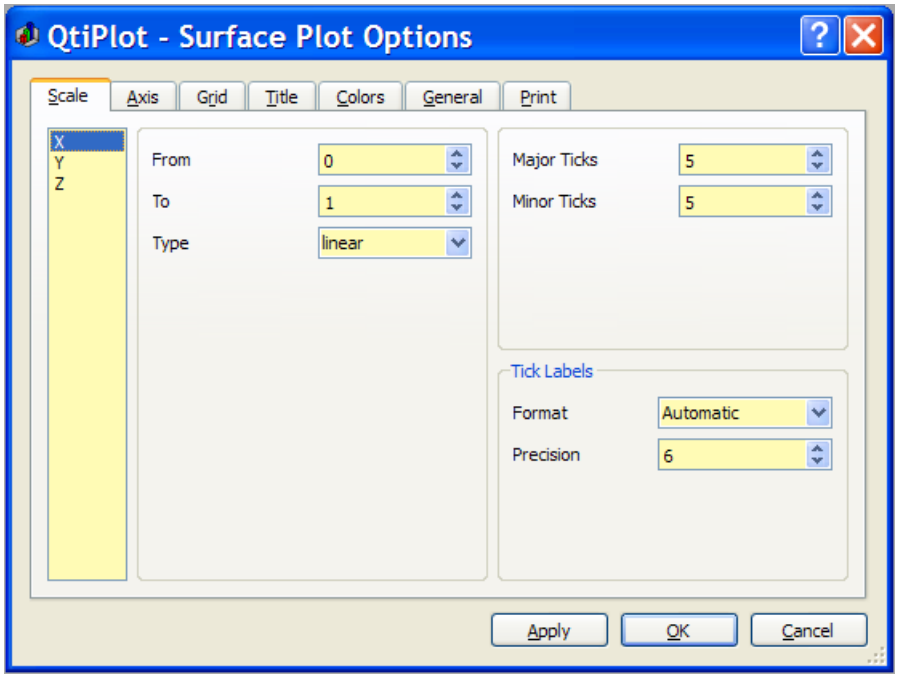
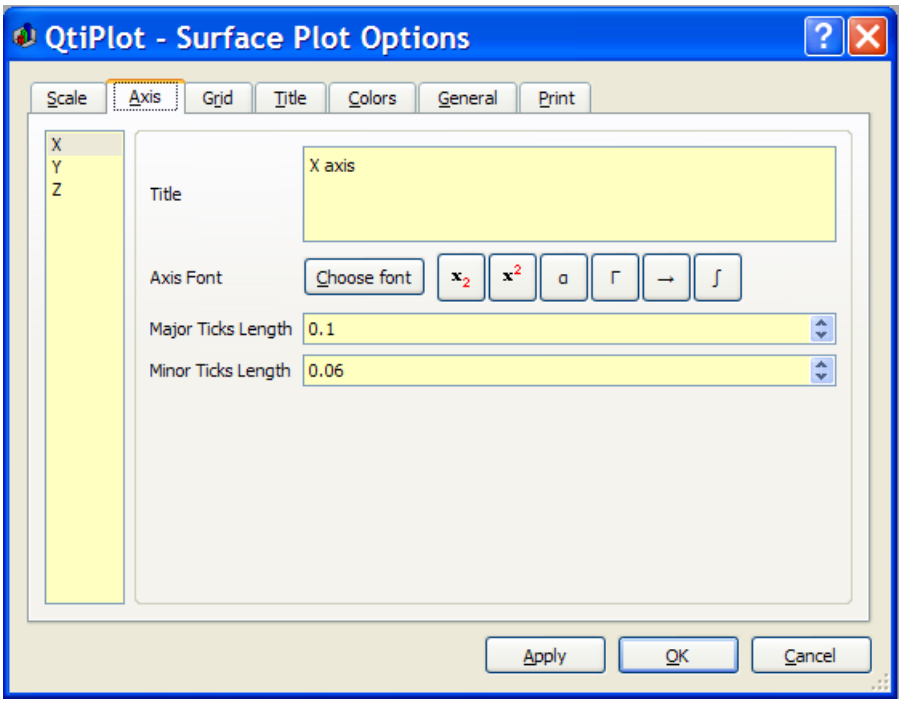


Figure 5.67: The surface plot options dialog box.

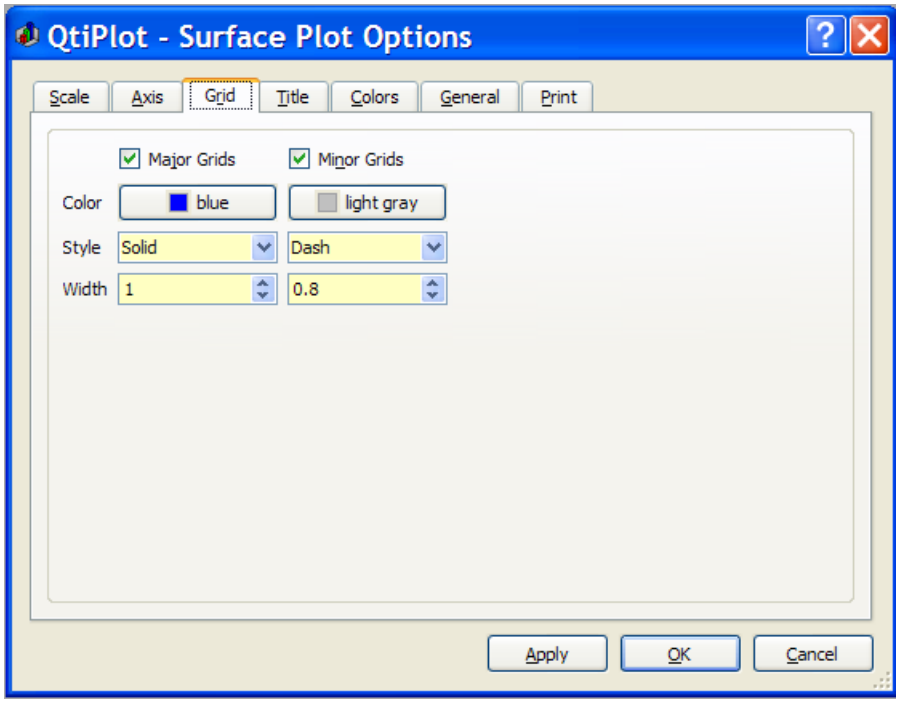
5.26.2 Axis Tab

The second tab defines the main parameters of the three axes: the axis label, its font, and the length of the ticks. Tick length is defined in the same units as the axis range. If the graph scales are changed, QtiPlot will re-calculate the length of the ticks. The font button on this tab only allows modification of the font used for the label. The font used for axis numbers is selected using the font control on the General Tab. The superscript and subscript buttons allow for 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, the exported LaTex file must be compiled separately, therefore you need a LaTeX environment installed on your computer.



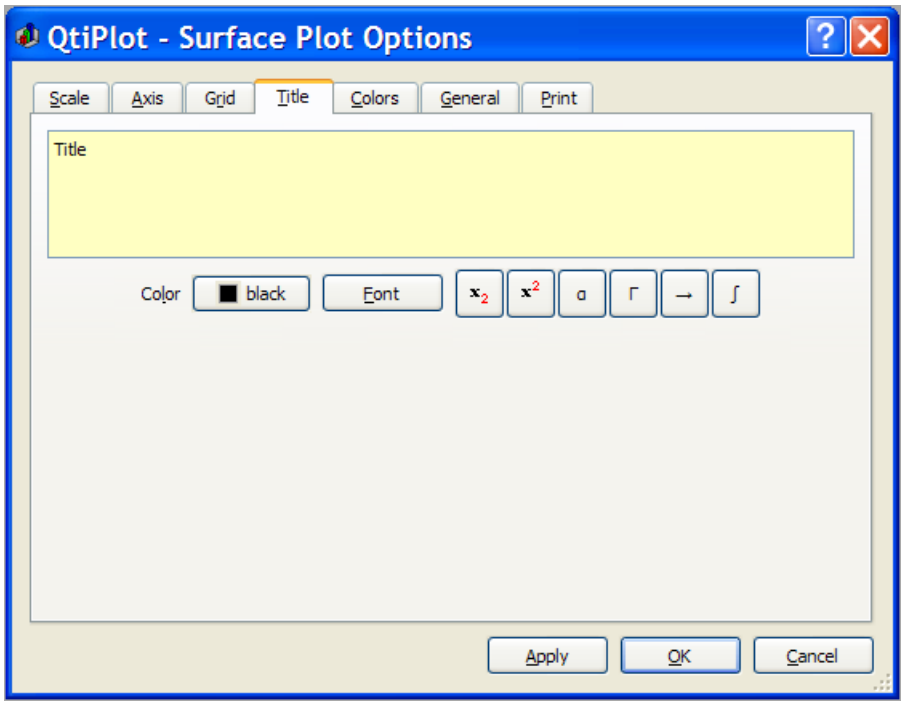
5.26.3 Grid Tab

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



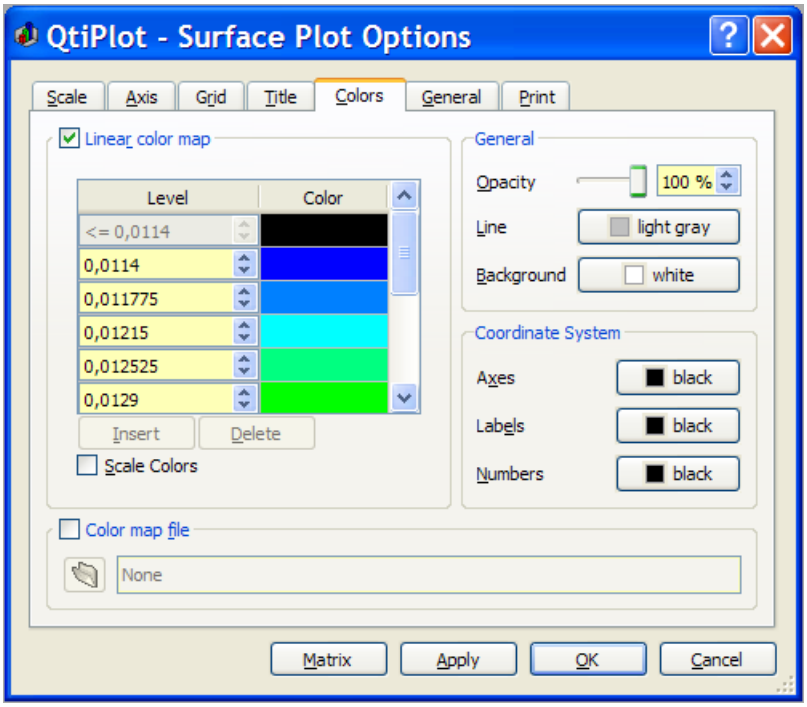
5.26.4 Title Tab

The fourth tab is used to define or modify the title of the plot. You can add LaTex format subscripts, superscripts, bold characters, etc in your title in a manner identical to that described for LaTex formatting in the Axis Tab.



5.26.5 Colors Tab

The fifth tab allows modification of the colors used for different elements of the plot.



The *Linear color map* group defines the color scheme which is used to show 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.

You may also 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. Numbers should be separated by spaces. You can find several examples of colormaps on the QtiPlot web site.

5.26.6 General Tab

The *General* tab is used to define some global parameters and the aspect ratio of the plot. The default behavior is to use perspective to compute the 3D plot. If you choose to check the *Orthogonal* check box, the plot will use a vertical Z axis regardless of the view angle of the plot.

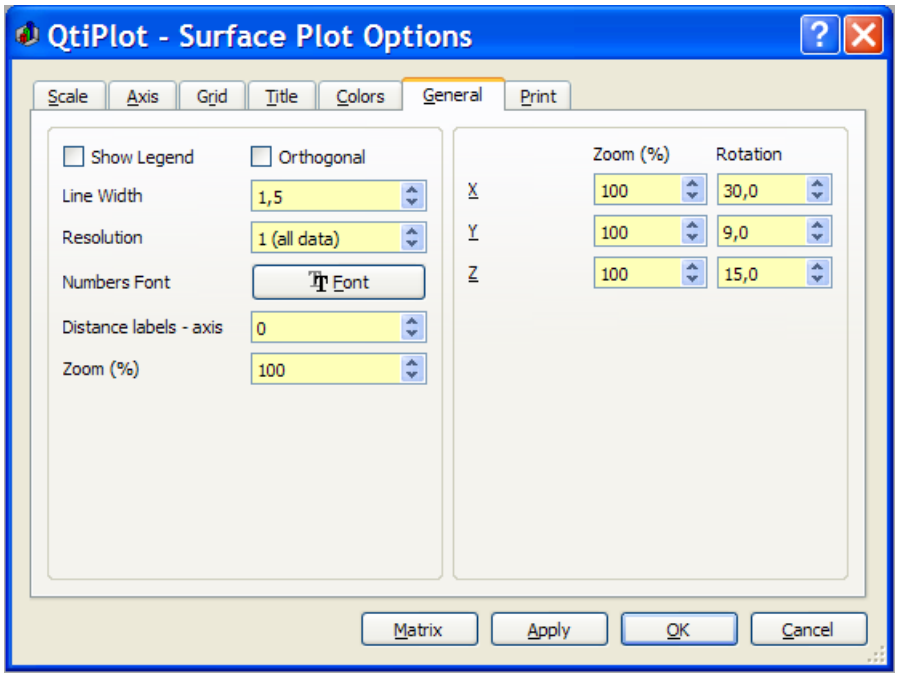


Figure 5.68: The general plot options tab.

5.26.7 Print Tab

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

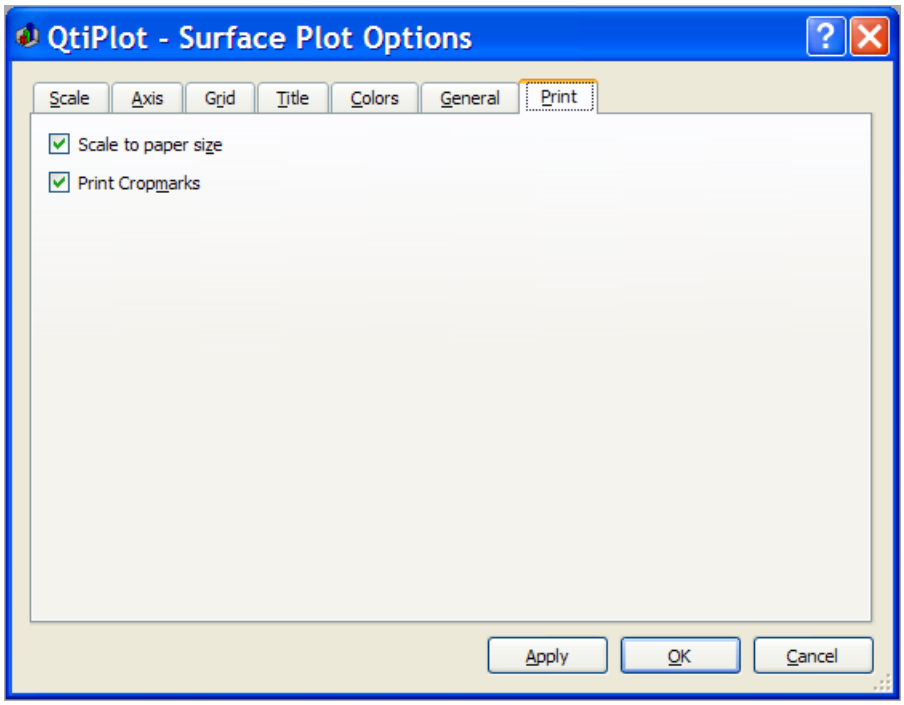


Figure 5.69: The 3D plot print options.

5.27 Text options

This dialog can be opened using several different commands, such as the Title... command, or simply by double clicking 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.

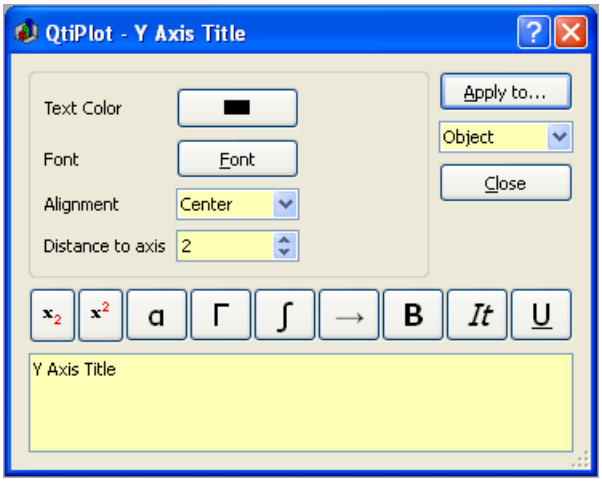


Figure 5.70: The axis title options dialog.

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 adding text boxes to a plot layer. When the text object is a legend (a listing of 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 following syntax 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 alternatively use the name or comment of the dataset, e.g. *\l(2)%(2,@C)* will use the column name. Additionally, 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 these format options to all text objects in the active plot layer, the current plot window, or all plot windows in the project. The scope of application of the new format can be chosen from the list box below the *Apply to...* button.

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

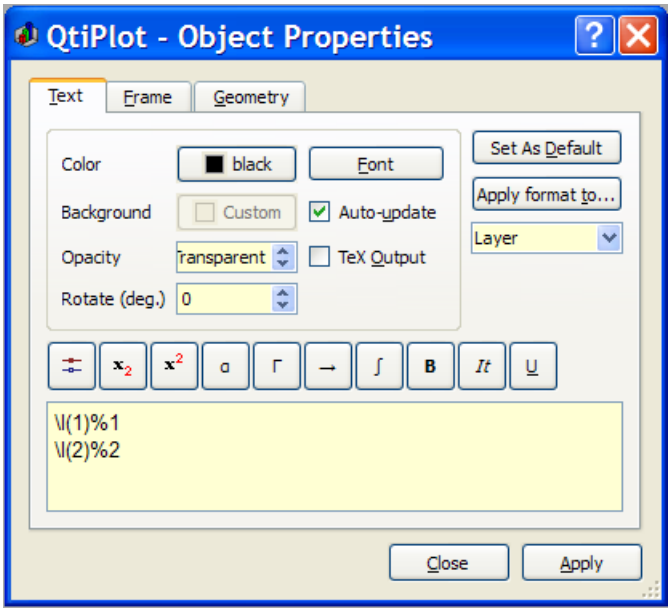
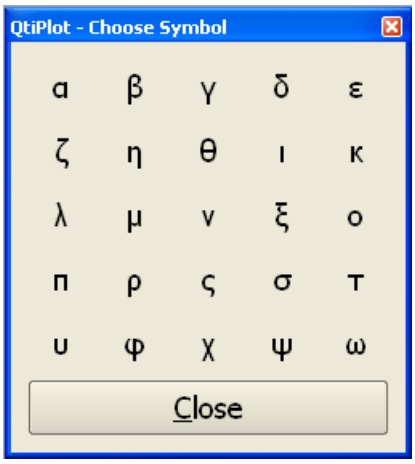


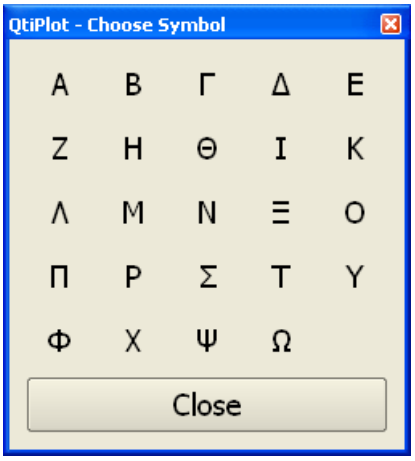
Figure 5.71: The legend/text options dialog.

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

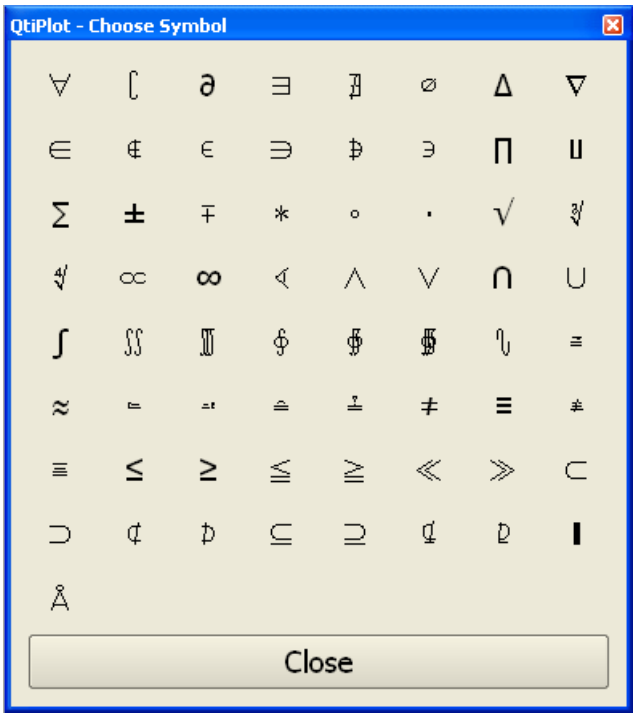
- _{text} will draw the text as subscripts. You can insert this sequence by clicking on the button.
- ^{text} will draw the text as superscripts. You can insert this sequence by clicking on the button.
- Clicking on the button, opens a dialog for selecting lower case Greek characters:



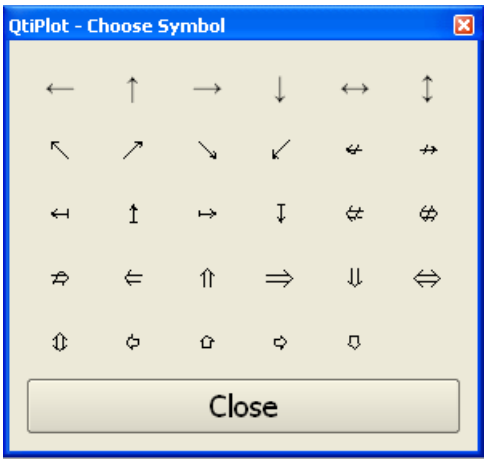
- Clicking on the Γ button opens a dialog for selecting upper case Greek characters:



- Clicking on the integral symbol opens a dialog which allows selecting various mathematical symbols:



- Clicking on the arrow icon opens a dialog which allows selecting various arrow symbols:



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

Chapter 6

Analysis of data and curves

6.1 Fast Fourier Transform

This function is accessed with the FFT... command, which is located in the Analysis Menu whenever a table or a plot is selected. The Fourier transform decomposes a signal into its elementary components by assuming that the signal x(t) can be described as a sum:

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

EQUATION 6.1: Fourier equation

in which ω_n are the frequencies, a_n are the amplitudes of each frequency and ψ_n are the phases of each frequency. QtiPlot will compute these parameters and build a new plot of the amplitude as a function of frequency.

This FFT was performed on a curve to extract it's characteristic frequency components. The original 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.

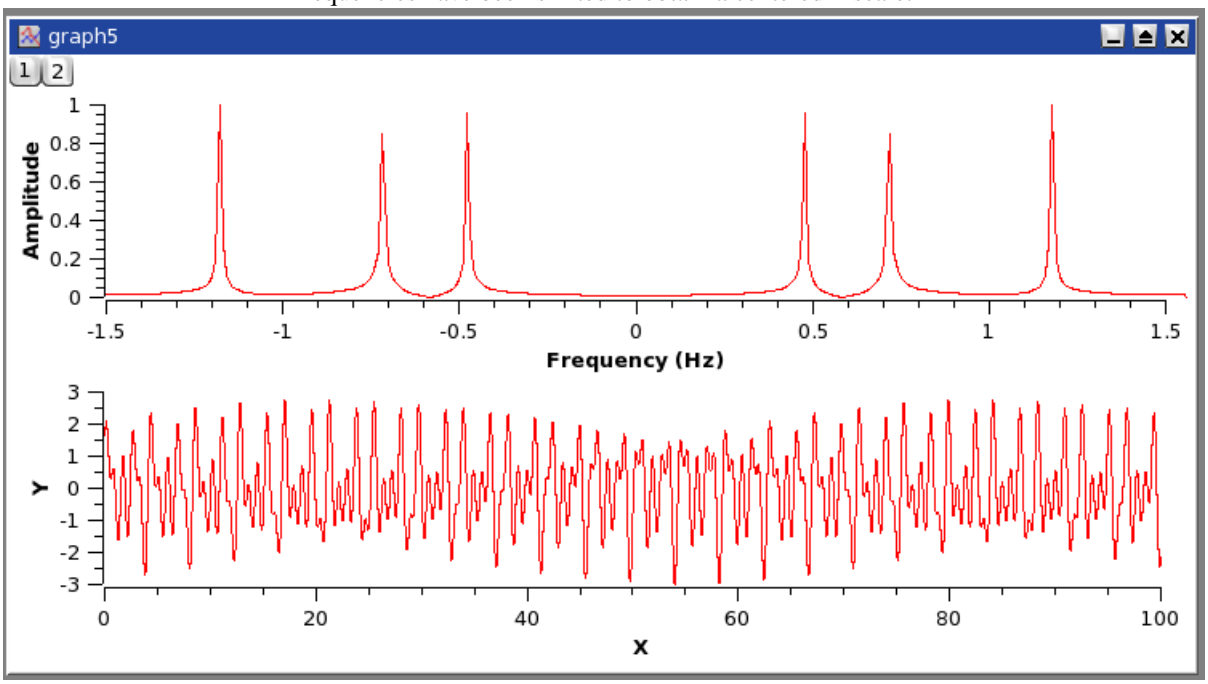


Figure 6.1: An example of the FFT.

The important parameters of the FFT can be modified using the FFT dialog, including the selection of an inverse FFT. An inverse FFT performed on the results of a forward FFT should result in the original signal. Frequently, it is useful to remove or modify some of the frequency components before the inverse FFT is performed. This may be particularly useful when it is desirable to remove some known interference. A common example would be the removal of power-line interference (usually 50 or 60 Hz, depending upon where you live). You should remember that this will also remove any real portions of the signal that happen to be at that same frequency. Care should be used when doing this.

6.2 Correlation

This function is accessed with the Correlate command. It is located in the Analysis Menu whenever 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 as:

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

EQUATION 6.2: 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 indexes 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 characteristic frequencies of the correlation function.

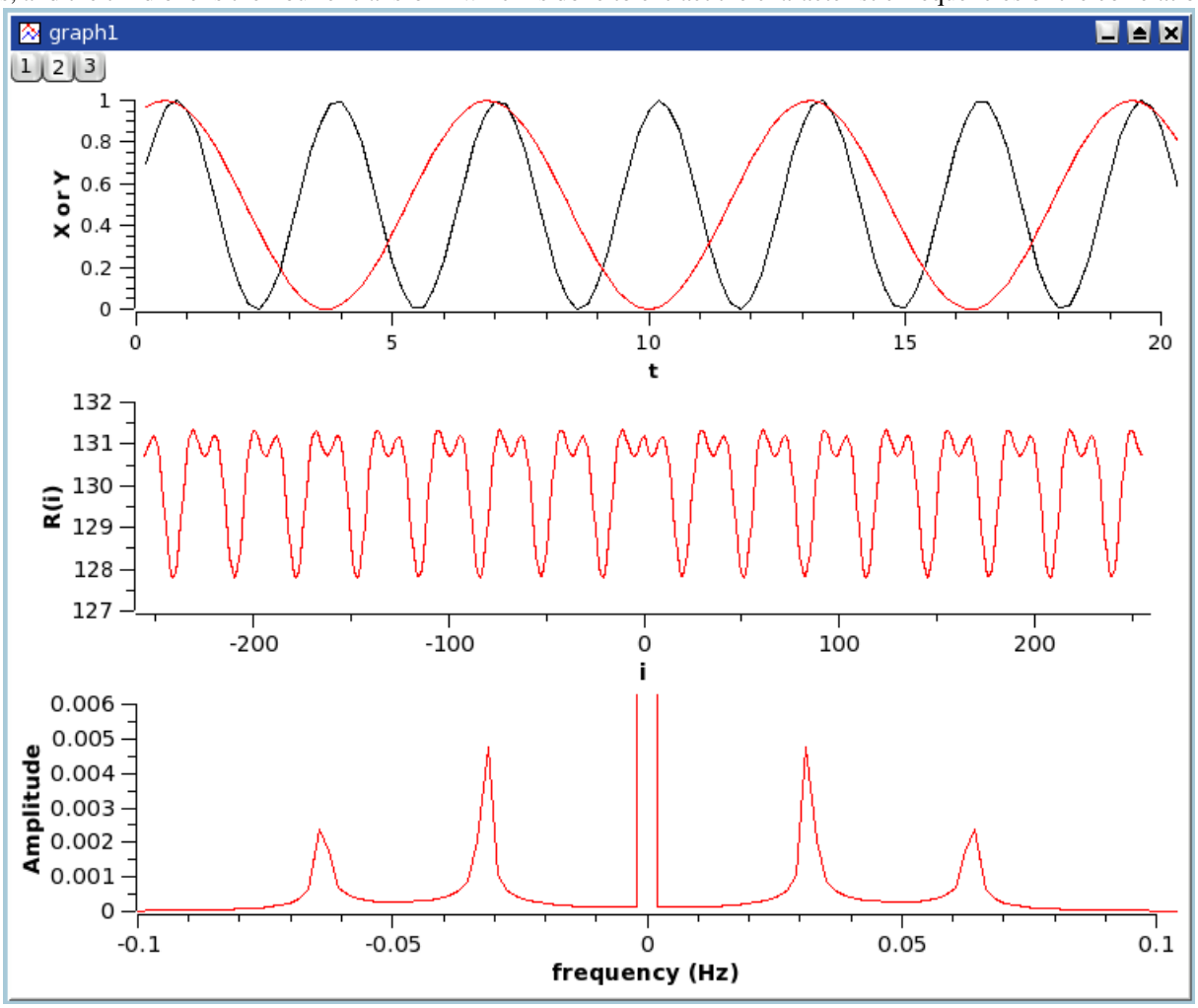


Figure 6.2: An example of a correlation between two sine functions.

The correlation of a signal with itself can also be performed and is frequently 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 is accessed with the Fit Wizard... command. It is located in the Analysis Menu whenever 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.

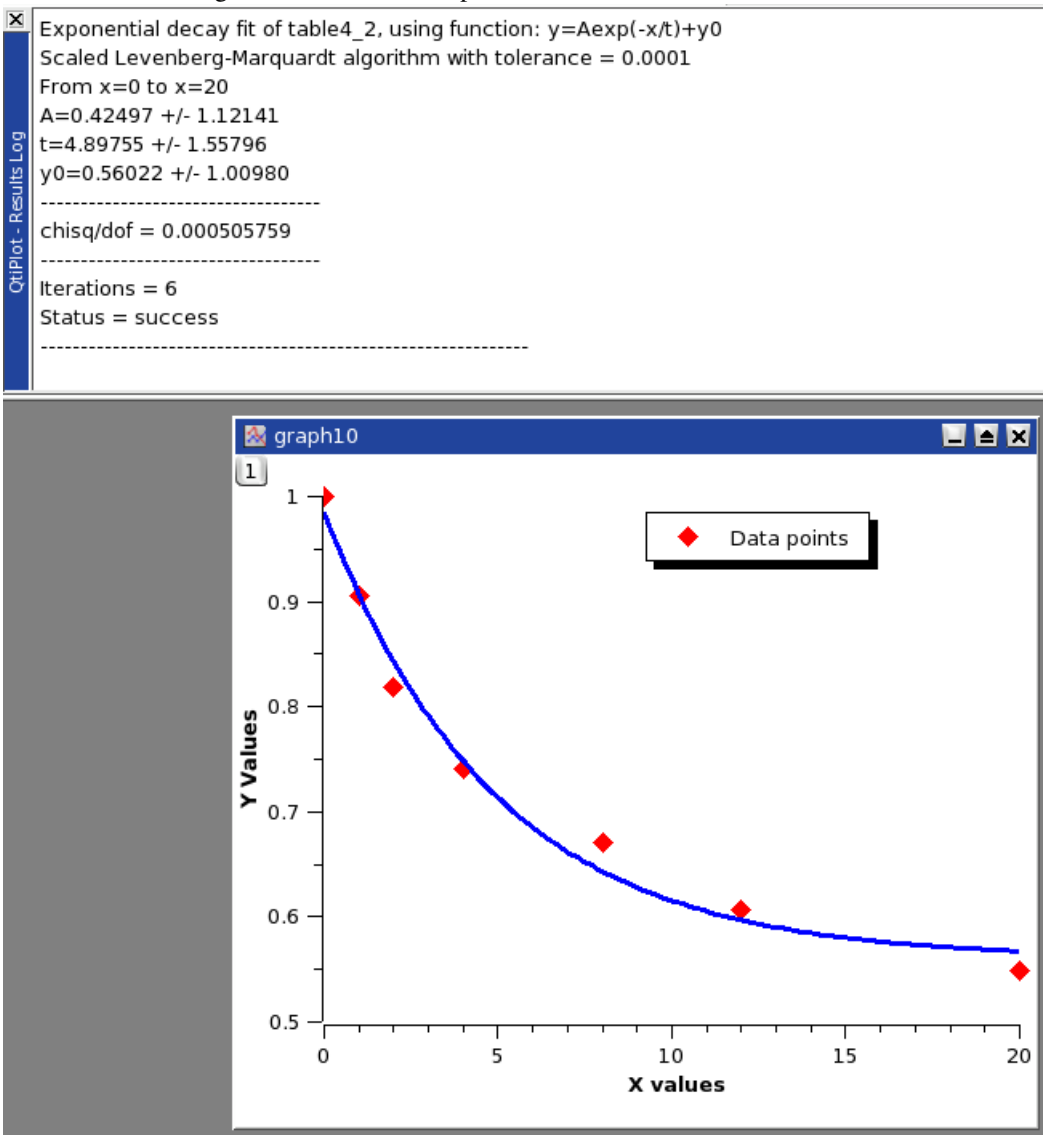


Figure 6.3: The results of the **Fit Wizard...**.

6.6 Fitting to specific curves

QtiPlot includes quick access to the most useful functions for fitting.

6.6.1 Fitting to a line

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

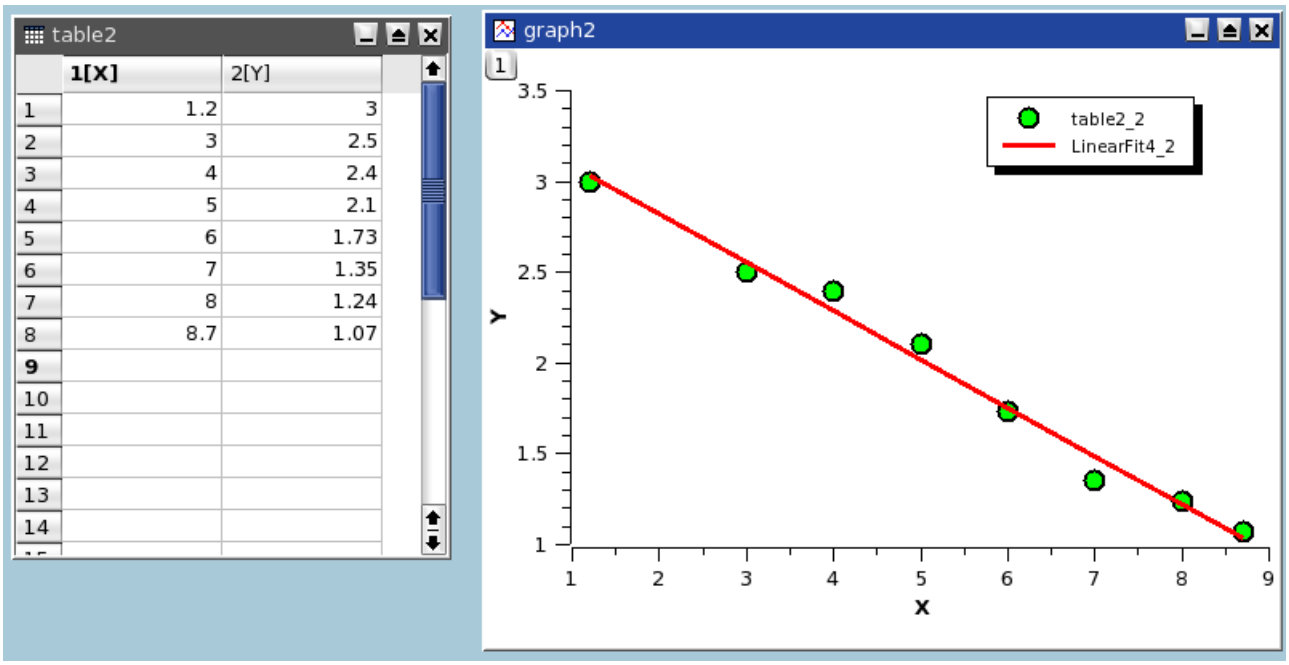
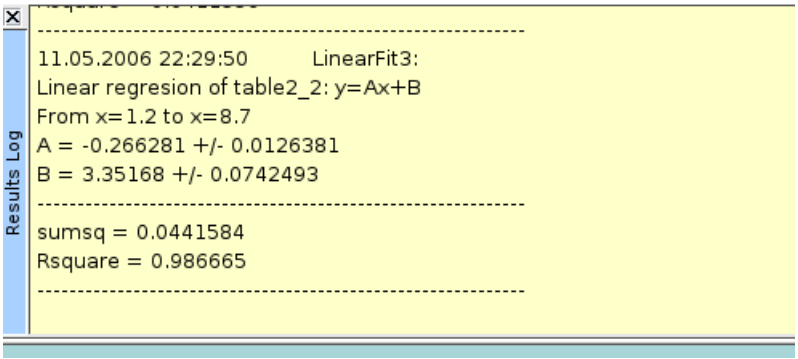


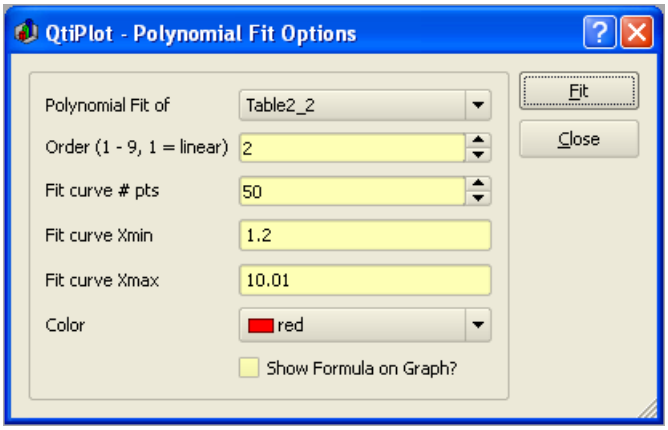
Figure 6.4: The results of a **Fit Linear**.

The results will be given in the Log panel:



6.6.2 Fitting to a polynomial

This command is used to fit a polynomial function to data which has a curvilinear shape. Results are be given in the Log panel



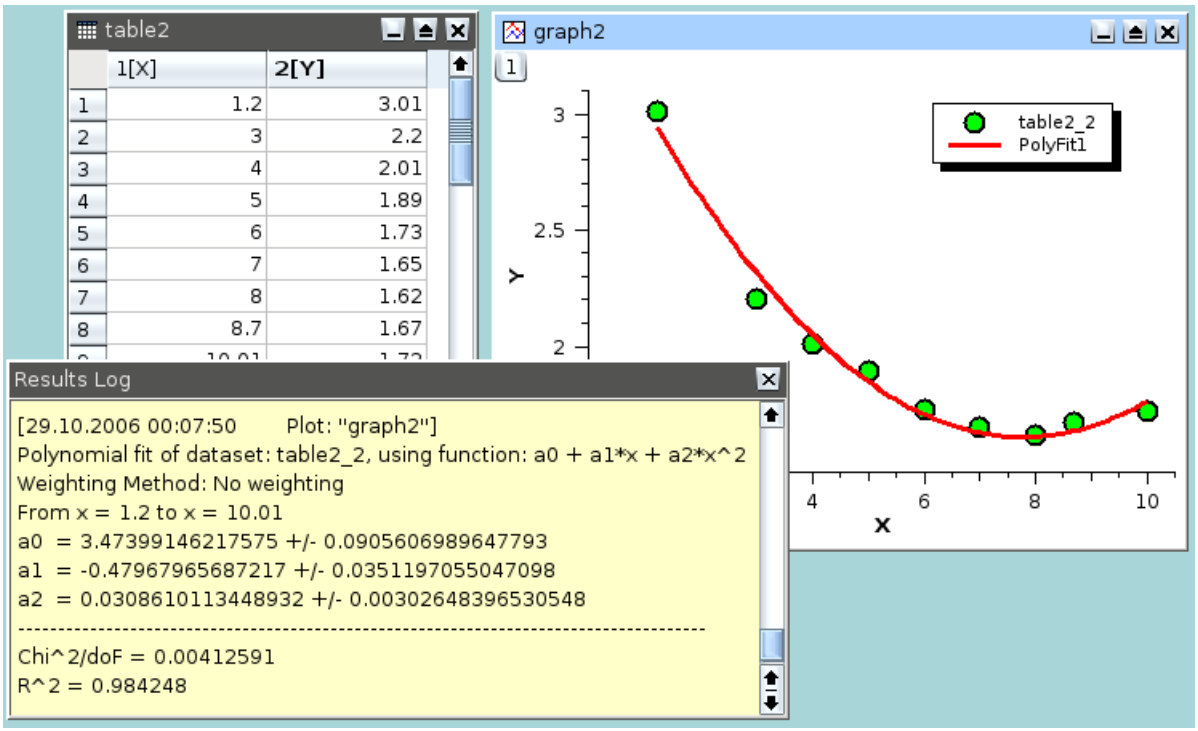


Figure 6.5: The results of a **Fit Polynomial...**, showing the initial data, the curve added to the plot, and the results in the log panel.

6.6.3 Fitting to a Boltzmann 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 6.3: Boltzmann equation

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

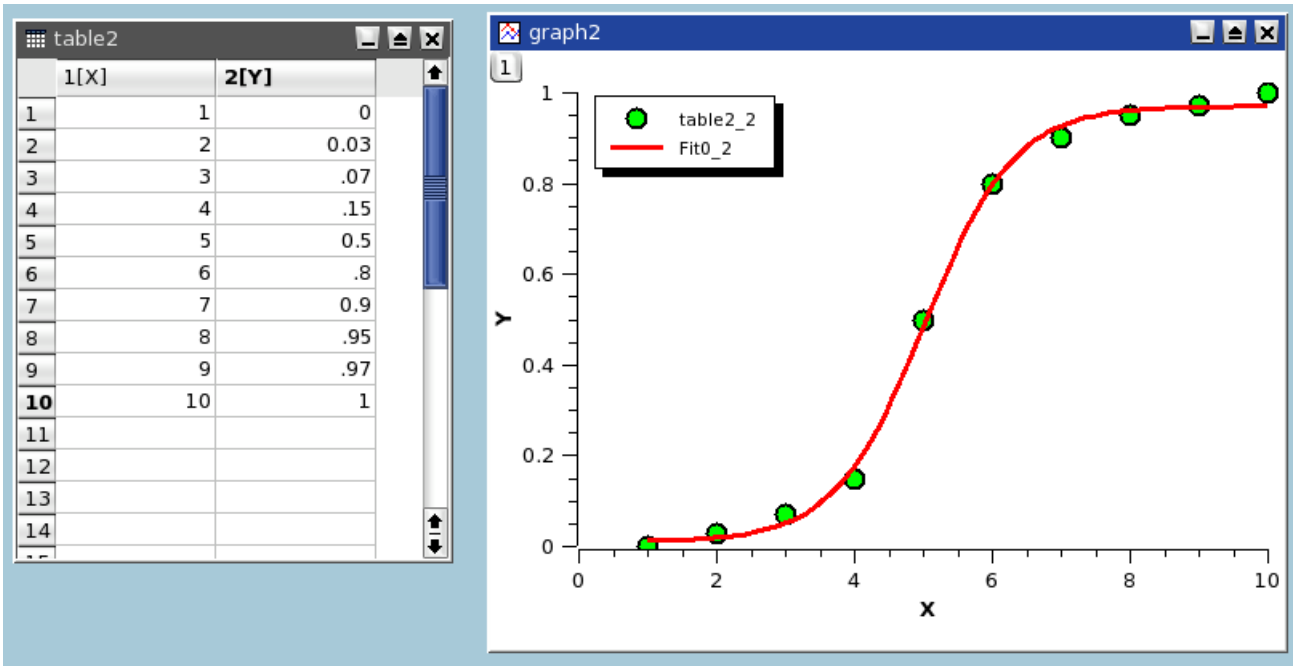


Figure 6.6: The results of a **Fit Boltzmann (sigmoidal)**.

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

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

EQUATION 6.4: 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 6.5: 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.

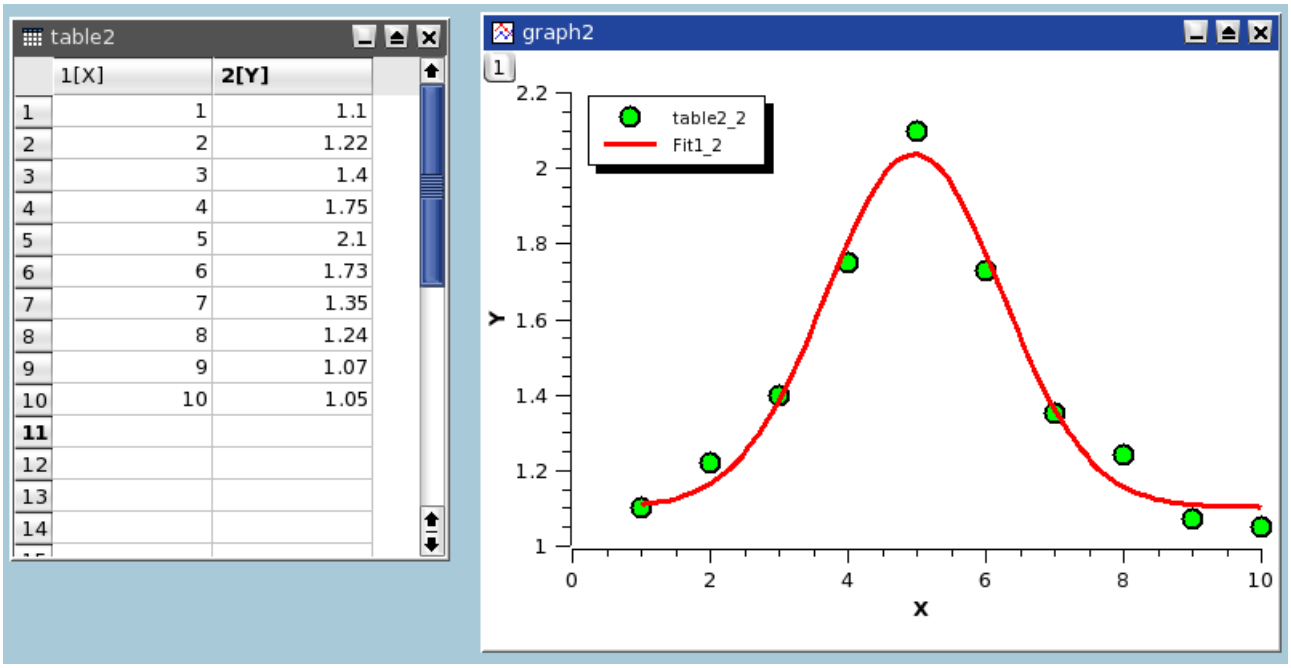


Figure 6.7: The results of a **Fit Gaussian**.

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:

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

EQUATION 6.6: 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.

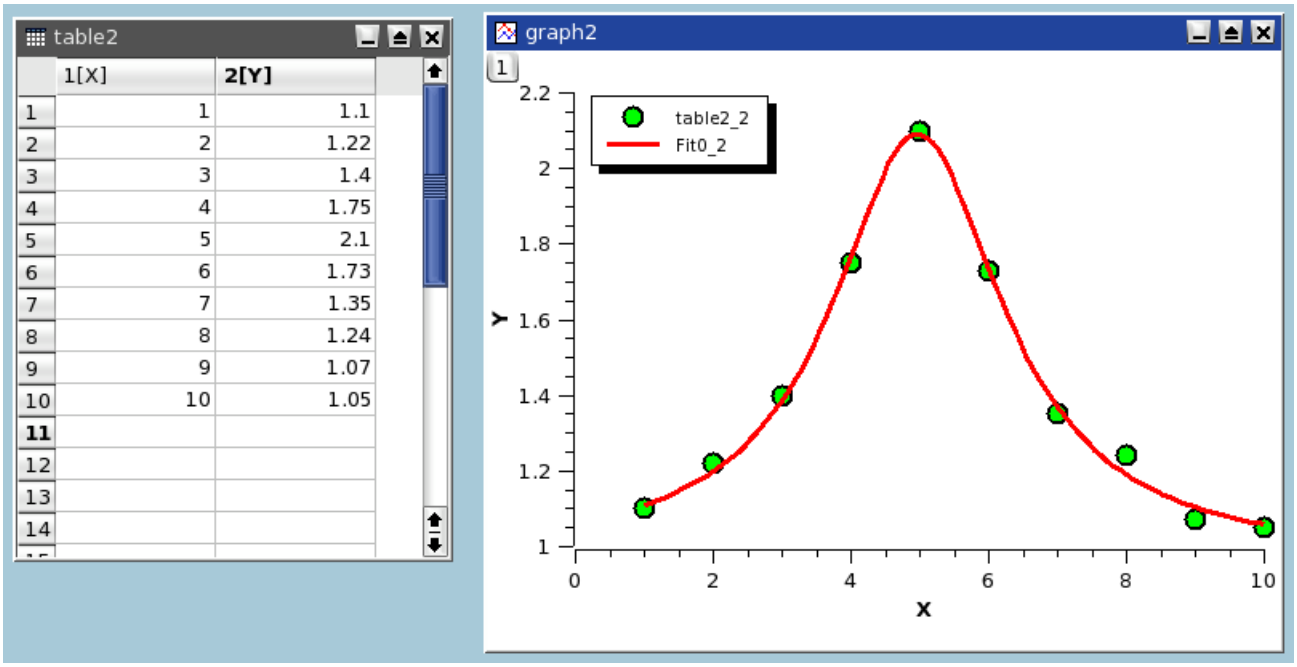


Figure 6.8: The results of a **Fit Lorentzian**.

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.

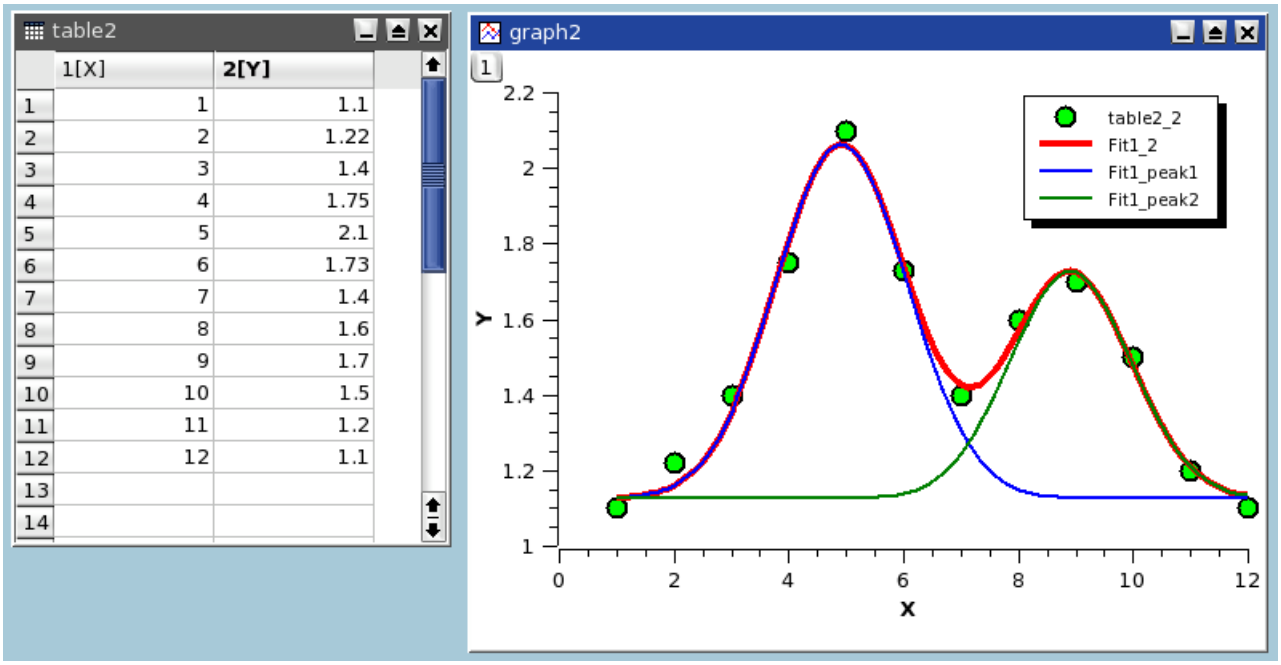
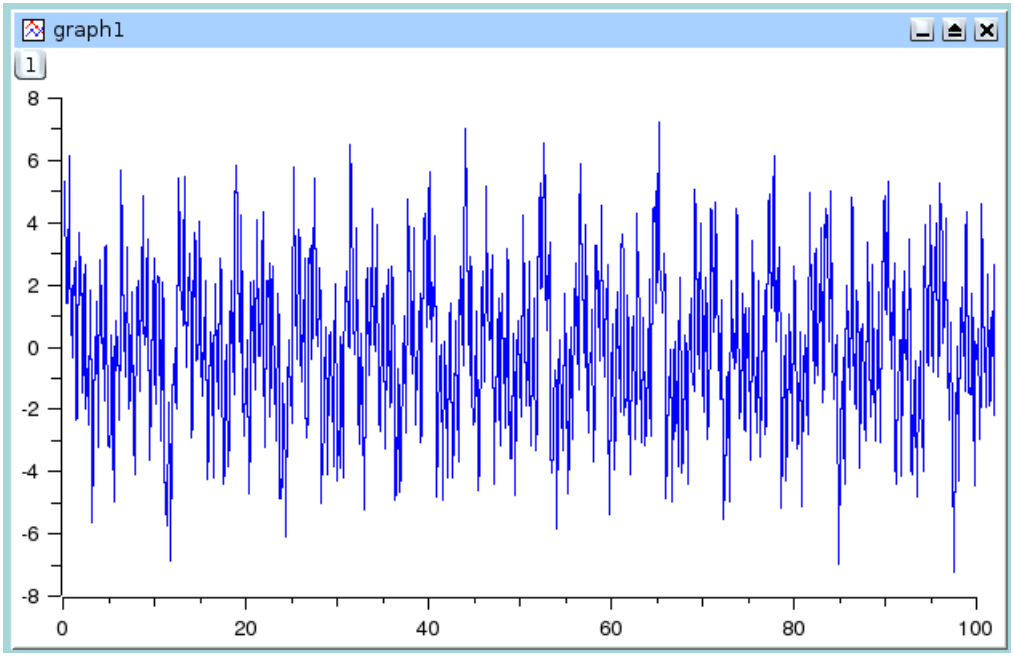


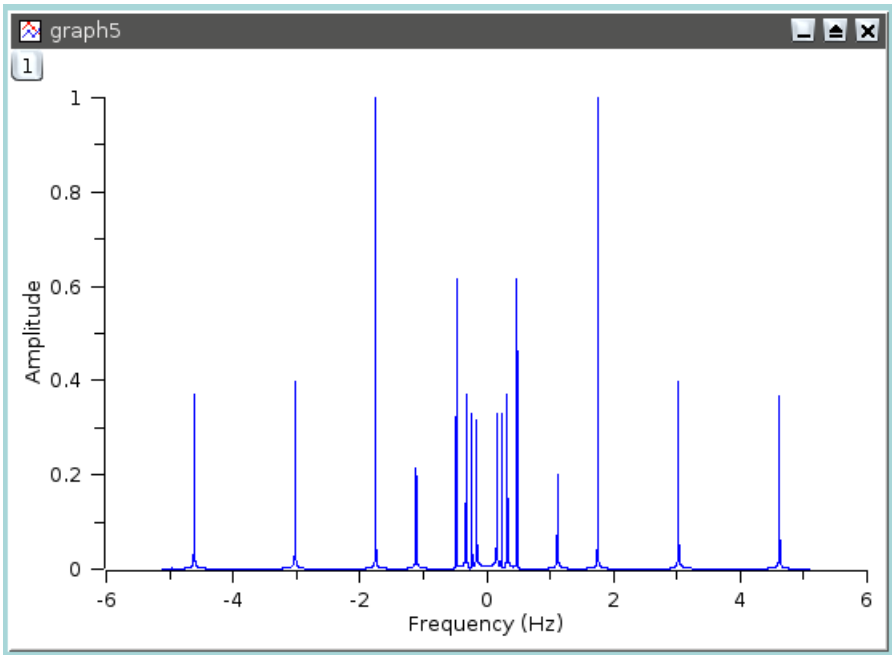
Figure 6.9: The results of a **Fit Multi-peak ->Gaussian...**.

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 which contains both high and low frequencies. We can analyze this by doing a FFT on the data curve, this leads to the following figure:



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

6.8.1 FFT low pass filter

This filter cuts high frequencies from 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:

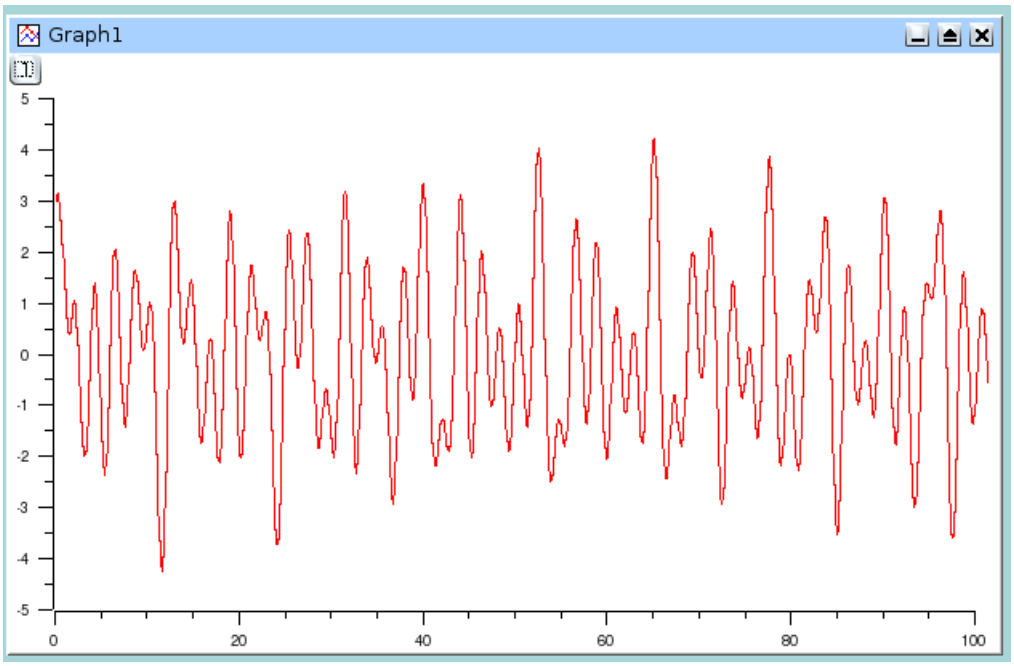
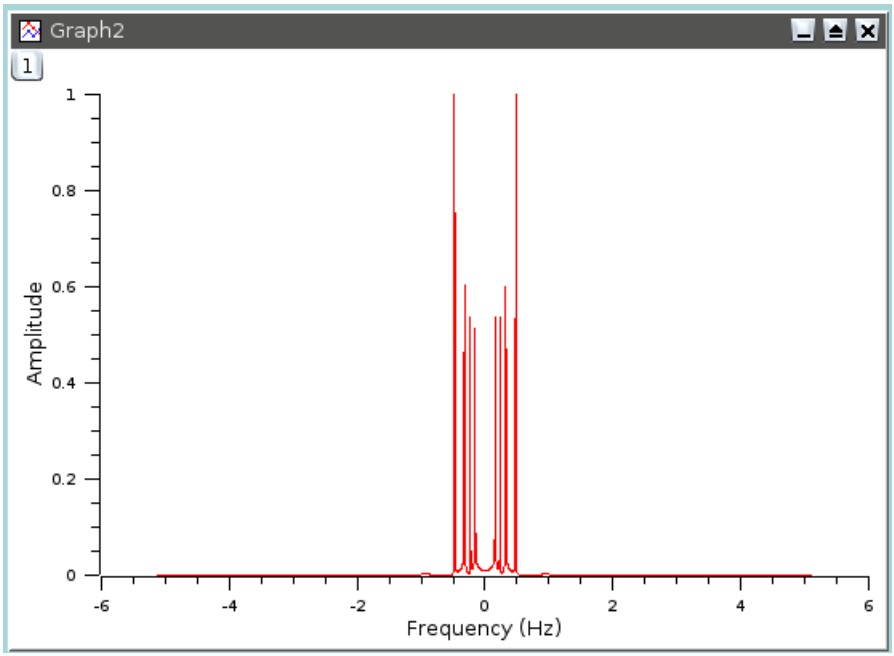


Figure 6.10: Signal after a FFT low pass filter

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



6.8.2 FFT high pass filter

This filter cuts low frequencies from 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:

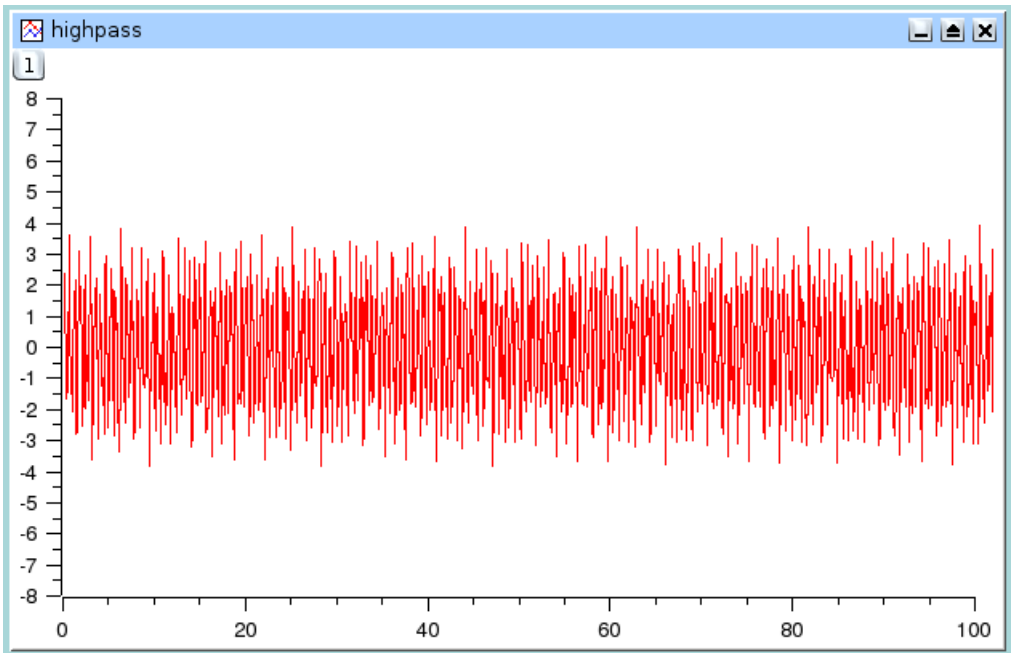
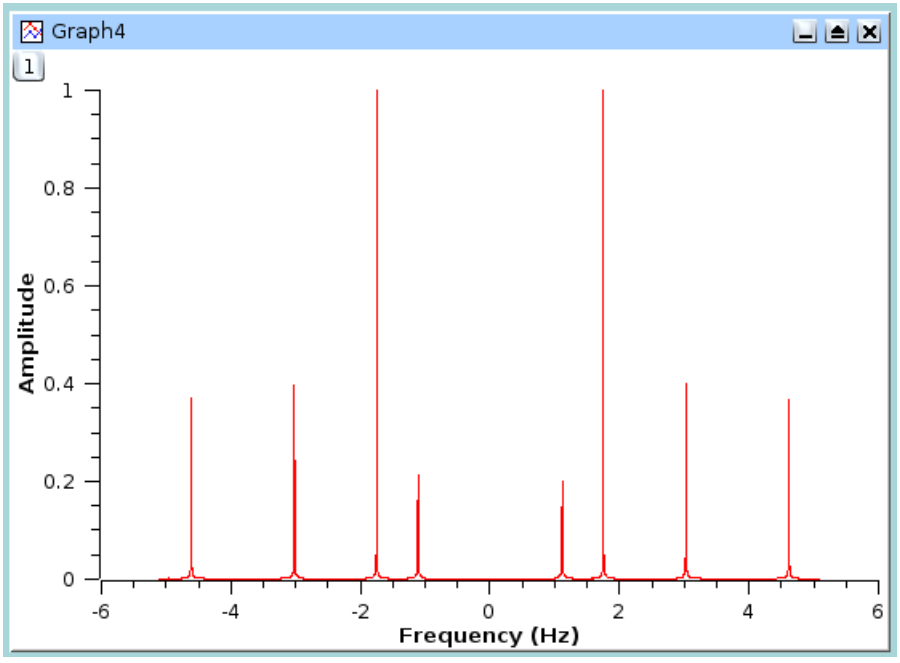


Figure 6.11: Signal after a FFT high pass filter

The power spectrum of this new signal shows that the frequencies above 1 Hz remain.



6.8.3 FFT band pass filter

This filter cuts both low and high frequencies from a signal while leaving frequencies in the pass band. 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:

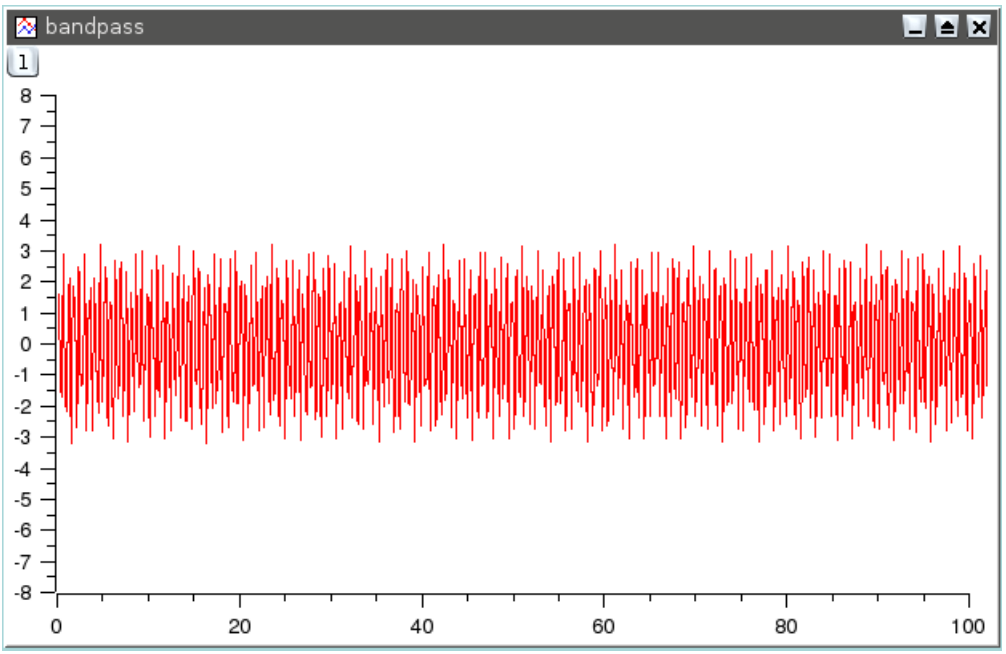
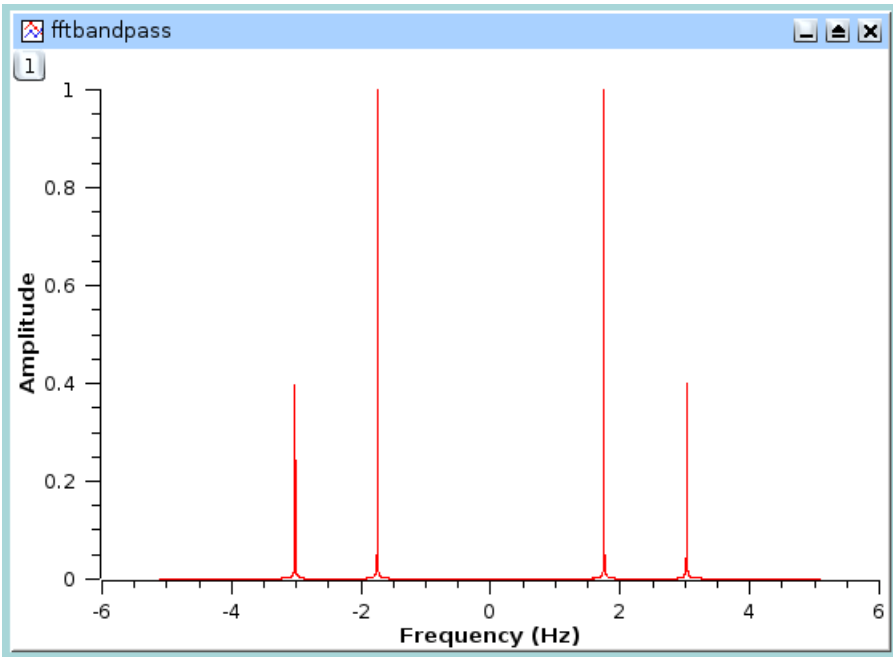


Figure 6.12: Signal after a FFT band pass filter

The power spectrum of this new signal shows that only frequencies in the pass band remain (2 and 3 Hz).



6.8.4 FFT block band filter

This filter keeps the low and high frequencies in a signal while cutting frequencies in the stop band. 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:

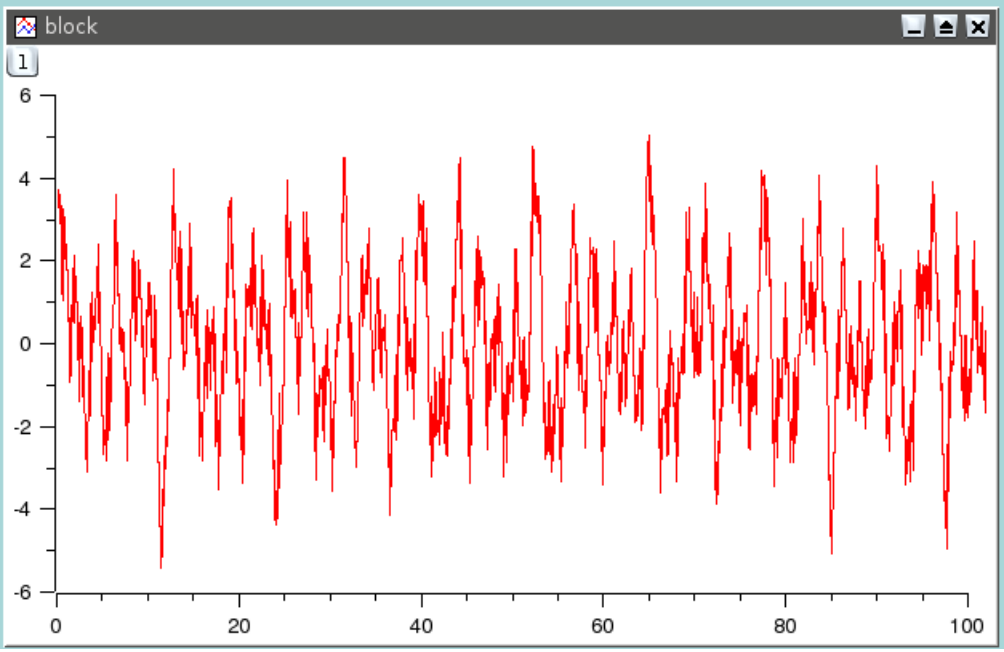
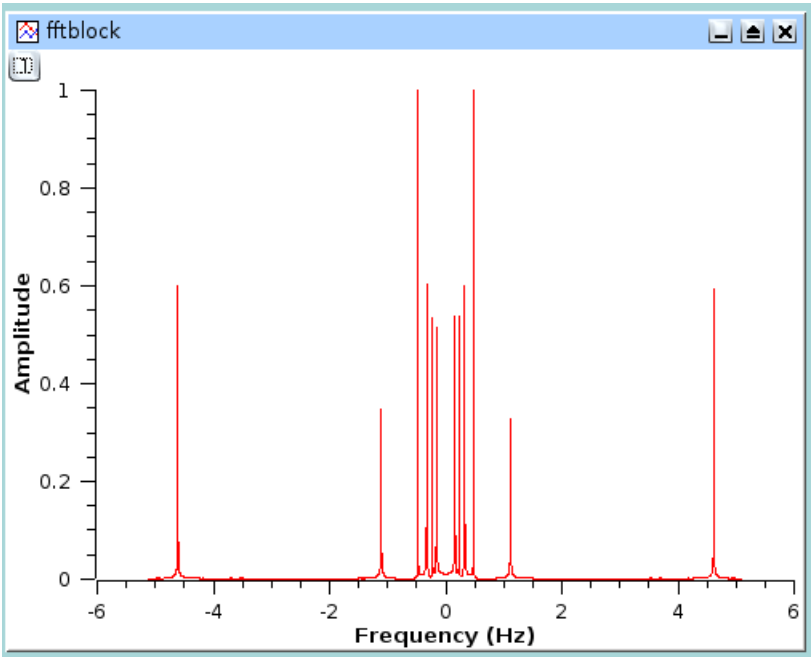


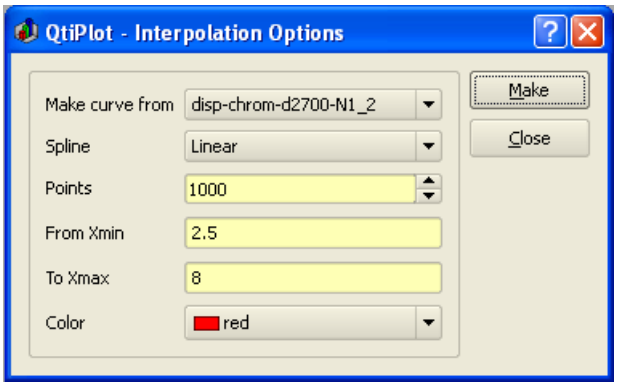
Figure 6.13: Signal after a FFT block band filter

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



6.9 Interpolation

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



Three interpolation methods are available. The first is a linear method. In this case, interpolated data points are placed along the straight between every two adjacent values of your data. The second is the method of cubic splines, in which case at least 4 points are needed. The last method, *Akima*, is a polynomial interpolation. Refer to the corresponding sections of the GNU Scientific Library for more details.

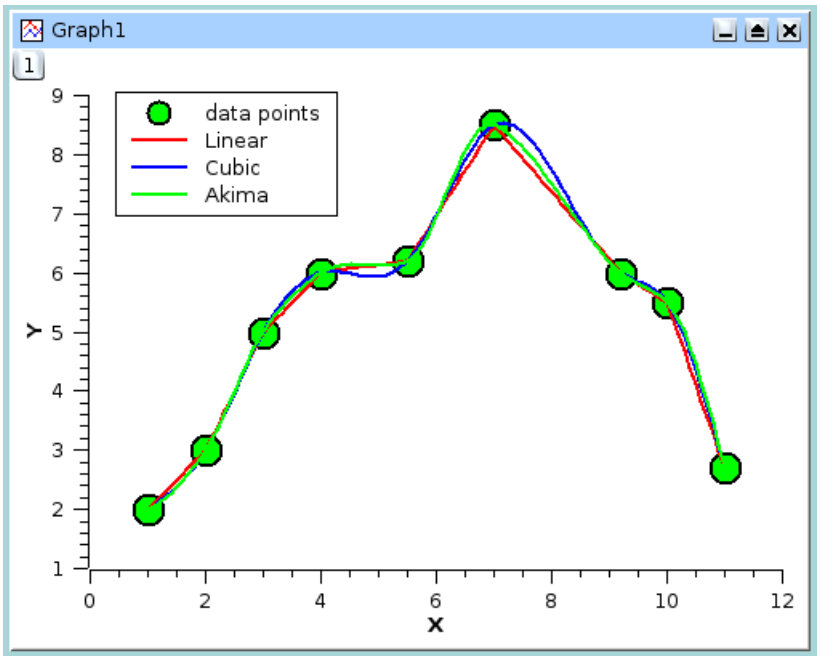


Figure 6.14: Comparison of the three methods of interpolation

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: muParser: Predefined Fundamental Physical Constants

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

Name	Description
+	Addition
-	Subtraction
*	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: muParser: Supported Mathematical Operators

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).

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. All statements outside of any function use module scope; that is, variables

Name	Description
abs(x)	absolute value of x
acos(x)	inverse cosine
acosh(x)	inverse hyperbolic cosine
asin(x)	inverse sine
asinh(x)	inverse hyperbolic sine
atan(x)	inverse tangent
atanh(x)	inverse hyperbolic tangent
avg(x1,x2,x3,...)	average value, this command accept a list of arguments separated by commas
bessel_j0(x)	Regular cylindrical Bessel function of zeroth order, $J_0(x)$.
bessel_j1(x)	Regular cylindrical Bessel function of first order, $J_1(x)$.
bessel_jn(x,n)	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(x,n)	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)	cosine of x
cosh(x)	hyperbolic cosine of x
erf(x)	error function of x
erfc(x)	Complementary error function erfc(x) = 1 - erf(x).
erfz(x)	The Gaussian probability density function Z(x).
erfq(x)	The upper tail of the Gaussian probability function Q(x).
exp(x)	Exponential function: e raised to the power of x.
gamma(x)	Computes the Gamma function, subject to x not being a negative integer
gammaln(x)	Computes the logarithm of the Gamma function, subject to x not a being negative integer. For x<0, log(|Gamma(x)|) is returned.
hazard(x)	Computes the hazard function for the normal distribution h(x) = erfz(x)/erfq(x).
ln(x)	natural logarithm of x
log(x)	decimal logarithm of x
log2(x)	base 2 logarithm of x
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)	sine of x
sinh(x)	hyperbolic sine of x
sqrt(x)	square root of x
tan(x)	tangent of x
tanh(x)	hyperbolic tangent of x

Table 7.3: muParser: Mathematical Functions

Name	Description
cell(a,b)	In the context of a matrix, returns the value at row a and column b. In the context of a table, returns the value at column a and row b (remember that tables use column logic). Everywhere else, this function is undefined.
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: muParser: Non-Mathematical Functions

attached to one column script or script window. All statements inside a function use that function's "local" scope, but they can also read module variables. System-wide globals are stored in the special variable `globals`. To store your own:

```
globals.mydata = 5
print globals.mydata
```

Global scope rules recently changed In older versions, write this instead:

```
global mydata
mydata = 5
print mydata
```

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 other modules, you have to add it to the `globals`:

```
def answer():
  return 42
globals.answer = answer
```

If you have an older versions of QtiPlot, 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:

```
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"]
```

Global scope rules recently changed In older versions, a function's local scope hid the module scope. That means that if you entered a function definition in a Note, you would not be able to access (neither reading nor writing) Note-local variables from within the function. However, you could 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 cosine
asin(x)	inverse sine
atan(x)	inverse tangent
atan2(y,x)	equivalent to atan(y/x), but more efficient
ceil(x)	ceiling; smallest integer greater or equal to x
cos(x)	cosine of x
cosh(x)	hyperbolic cosine of x
degrees(x)	convert angle from radians to degrees
exp(x)	Exponential function: e raised to the power of x.
fabs(x)	absolute value of x
floor(x)	largest integer smaller or equal to x
fmod(x,y)	remainder of integer division x/y
frexp(x)	Returns the tuple (mantissa,exponent) such that x=mantissa*(2**exponent) where exponent is an integer and 0.5 <=abs(m)<1.0
hypot(x,y)	equivalent to sqrt(x*x+y*y)
ldexp(x,y)	equivalent to x*(2**y)
log(x)	natural (base e) logarithm of x
log10(x)	decimal (base 10) logarithm of x
modf(x)	return fractional and integer part of x as a tuple
pow(x,y)	x to the power of y; equivalent to x**y
radians(x)	convert angle from degrees to radians
sin(x)	sine of x
sinh(x)	hyperbolic sine of x
sqrt(x)	square root of x
tan(x)	tangent of x
tanh(x)	hyperbolic tangent of x

Table 7.5: Python: 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 straightforward,

```
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()
# display some information in the data display toolbar:
displayInfo(text)
# get a pointer to the QLineEdit object in the data display toolbar and access the ↩
    information displayed:
info = infoLineEdit()
text = info.text()
```

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 following 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 determined 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. Here's how you can access and modify the geometry of a window in the project:

```
t = table("Table1")
t.setGeometry(10, 10, 800, 600)
print t.x(), t.y(), t.width(), t.height()
```

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 analyzing 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 below:

```
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 iterable. The data is returned row wise as tuple.

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

It is possible to assign random or normal random values to a complete column or to a subset of rows in a column:

```
  t.setRandomValues(col, startRow = 1, endRow = -1)
  t.setNormalRandomValues(col, startRow = 1, endRow = -1, standardDeviation = 1.0)
```

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 column or a range of rows using the functions below:

```
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 being 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 character. 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 table 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.newTabl`** 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)
```

You can change the dimensions of a matrix:

```
m.setDimensions(rows, columns)
m.setNumRows(rows)
m.setNumCols(columns)
```

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

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

If QtiPlot has been built with support for ALGLIB, you can also change the dimensions of a matrix by resampling it, using bilinear or bicubic interpolation:

```
m.resample(rows, cols)# bilinear interpolation by default
m.resample(rows, cols, 1) # bicubic interpolation
```

If ALGLIB is available you can also smooth the matrix data using:

```
m.smooth()
```

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 process by simply converting the table into a matrix:

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

For the production of contour plots, you can convert a regular XYZ data table ("regular", meaning that cells in the X and Y columns of the table define a regular 2D grid) into a matrix:

```
m = tableToMatrixRegularXYZ(table("Table1"), "Table1_3")
```

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, compression = 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 quality 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). The compression parameter can be 0 (no compression) or 1 (LZW) and is only effectif for .tif/.tiff images. It is neglected for all other raster image formats.

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 characters. 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 stem-plot (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 stem-plot 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 stem-plot. The following function returns a string of characters representing 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 stem-plot, 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 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>")
```

7.2.10.2 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 opposite 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 below 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)
```

7.2.10.3 The canvas

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

```
l.setCanvasFrame(2, QtGui.QColor("red"))
l.setCanvasColor(QtGui.QColor("lightGrey"))
l.setCanvasBackgroundImage("C:/qtiplot/qtiplot/qtiplot_logo.png")
```

The following access methods are available for the canvas background image:

```
pic = l.backgroundPixmap() # QPixmap
path = l.canvasBackgroundFileName()
```

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.

7.2.10.4 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)
```

7.2.10.5 Customizing the grid

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

```
l.showGrid(axis)
l.showGrid()
l.setGridOnTop(on = True, update = True) # draw grid on top of data
```

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 lines, whereas the `Y` letter indicates the horizontal ones. Also, the `Maj` word refers to the main grid lines and `Min` to the secondary grid.

7.2.10.6 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()
```

7.2.10.7 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)
```

7.2.10.8 Resizing layers

A layer can be resized using the methods below, 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 behavior 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 behavior and keep the size of the fonts unchanged:

```
l.setAutoscaleFonts(False)
```

7.2.10.9 Resizing the drawing area

The drawing area of a layer (the canvas) can be resized using the methods below, 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.

7.2.11 Working with 2D 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 functions:

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

If the table column used to create a plot curve has empty cells, you can choose whether the curve line is drawn connected across the missing data or not. This behavior can be specified using:

```
l.showMissingDataGap(on = True, replot = True)
```

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

```
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 below:

```
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). 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 below. 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 below:

```
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 below:

```
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()
```

7.2.11.1 Curve 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

It is also possible to define a custom image as the plot symbol for a curve:

```
g = newGraph().activeLayer()
c = g.addFunction("cos(x)", 0, 10, 20)
c.setSymbol(ImageSymbol("qtiplot/manual/html/icons/help.png"))
```

Here's a short script showing how to draw a custom plot symbol and assign it to a curve:

```
pix = QtGui.QPixmap(QtCore.QSize(11, 11))
pix.fill(Qt.transparent)
p = QtGui.QPainter(pix)
r = QtCore.QRect(0, 0, 10, 10)
p.drawEllipse(r)
p.setPen(QtGui.QPen(Qt.red))
p.drawLine(5, 0, 5, 10)
p.drawLine(0, 5, 10, 5)
p.end()

g = newGraph().activeLayer()
c = g.addFunction("sin(x)", 0, 10, 20)
c.setSymbol(ImageSymbol(pix))
```

7.2.12 2D Analytical Functions

You can also add analytical function curves to a plot layer:

```
f = l.addFunction("x*sin(x)", 0, 3*pi, points = 100)
f.setTitle("x*sin(x)")
f.setPen(Qt.green)
f.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")
```

It is possible to get a reference to an analytical function curve on the layer l using it's index, like shown below:

```
f = l.functionCurve(curveIndex)
```

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

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

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

If you need to access the values and names of a function's parameters, you have at your disposal the following methods:

```
i = f.parametersCount() # the number of parameters in your function formula
name = f.parameterName(index) # the name of the parameter of rang index as a QString
p1 = f.parameterValue(index) # the value of the parameter of rang index as a double
p2 = f.parameterValue(name) # the value of a parameter using its name string
```

The abscissae range for which the function is calculated/displayed, can be obtained/modified via the methods below:

```
x1 = f.startRange()
x2 = f.endRange()
f.setRange(0.5, 15.2)
```

7.2.13 Error Bars

Having a plot layer *l*, you can add error bars to a data curve *c*, named *curveName*, using the following methods:

```
err1 = l.addErrorBars(c, Table *t, QString errColName, int type = 1, double width = 1, int ↩
    capLength = 8, color = Qt.black, throughSymbol = True, minusSide = True, plusSide = True ↩
    )
err2 = l.addErrorBars(curveName, Table *t, QString errColName, int type = 1, double width = ↩
     1, int capLength = 8, color = Qt.black, throughSymbol = True, minusSide = True, ↩
    plusSide = True)
```

Each data curve, *c*, can have attached a list of error bars:

```
errors = c.errorBarsList()
```

The properties of an error bar curve can be accesses, via the following methods:

```
err = c.errorBarsList()[0]
for i in range(0, err.dataSize()):
  print err.errorValue(i)

err.capLength()
err.width()
err.color()
err.direction()
err.xErrors()
err.throughSymbol()
err.plusSide()
err.minusSide()
c = err.masterCurve() # reference to the master curve to which the error bars curve is attached.
err.detachFromMasterCurve() # equivalent to c.removeErrorBars(err)
```

... and can be modified, via the following methods:

```
err.setCapLength(12)
err.setWidth(3)
err.setColor(Qt.red)
err.setDirection(ErrorBarsCurve.Vertical)
err.setXErrors(True) # equivalent to err.setDirection(ErrorBarsCurve.Horizontal)
err.drawThroughSymbol(True)
err.drawPlusSide(True)
err.drawMinusSide(False)
err.setMasterCurve(c)
```

You can remove all error bars attached to a curve using:

```
c.clearErrorBars()
```

7.2.14 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 below:

```
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 below:

```
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()
```

7.2.15 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()
```

7.2.16 Box and whiskers plots

The following script shows how to create and customize a box and whiskers plot:

```
g = newGraph().activeLayer()
g.plotBox(table("Table1"), ["2", "3", "4"])
for i in range (0, g.numCurves()):
    box = g.boxCurve(i)
    if (box):
      box.setBrush(QtGui.QBrush(Qt.red, Qt.BDiagPattern))
      s = PlotSymbol()
      s.setPen(QtGui.QPen(Qt.blue, 2))
      s.setSize(QtCore.QSize(10, 10))
      box.setSymbol(s)
      box.setMeanStyle(PlotSymbol.Cross)
      box.setBoxStyle(BoxCurve.WindBox)
      box.setBoxWidth(80)
      box.setWhiskersRange(BoxCurve.SD) # standard deviation
```

The following functions give access to the statistics information for the data series displayed with a box and whiskers plot:

```
s = box.statistics() # returns information as a string
m = box.median()
q = box.quantile(f) # f must be a fraction between 0 and 1
```

For an exhaustive review of the methods giving access to the properties of a box plot, please consult the QtiPlot/Python API.

7.2.17 Pie Plots

The following script shows how to create and customize a pie plot:

```
g = newGraph().activeLayer()
pie = g.plotPie(table("Table1"), "2")
pie.setRadius(70)
pie.setViewAngle(40)
pie.setThickness(20)
pie.setStartAzimuth(45)
pie.setLabelsEdgeDistance(50)
pie.setCounterClockwise(True)
pie.setBrushStyle(Qt.Dense3Pattern)
pie.setFirstColor(3)
pie.setPen(QtGui.QPen(Qt.red, 2))
pie.setLabelValuesFormat(True)
```

For an exhaustive review of the methods giving access to the properties of a pie plot, please consult the QtiPlot/Python API.

7.2.18 Vector Plots

The following script shows how to create and customize a vector curve:

```
g = newGraph().activeLayer()
v = g.plotVectors(table("Table1"), ["1", "2", "3", "4"], Layer.VectXYAM)
v.setVectorPen(Qt.red)
v.fillArrowHead(False)
v.setHeadAngle(15)
v.setHeadLength(20)
v.setPosition(VectorCurve.Middle)
```

For an exhaustive review of the methods giving access to the properties of a vector curve, please consult the QtiPlot/Python API.

7.2.19 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 below:

```
g = graph("Graph1").activeLayer()
lst = g.arrowsList()
for i in range (0, g.numArrows()):
  lst[i].setColor(Qt.green)

g.replot()
```

7.2.20 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)
```

7.2.21 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)
```

7.2.22 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)
```

7.2.23 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 = QtCore.QSizeF(), unit = Frame.Pixel ↩
    , fontsFactor = 1.0)
l.exportImage(fileName, quality = 100, transparent = False, dpi = 0, size = QtCore.QSizeF() ↩
    , unit = Frame.Pixel, fontsFactor = 1.0, compression = 0)
l.exportTex(fileName, color = True, escapeStrings = True, fontSizes = True, size = QtCore. ↩
    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 raster 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

quality 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 `compression` parameter can be 0 (no compression) or 1 (LZW) and is only effectif for .tif/.tiff images. It is neglected for all other raster image formats.

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.

7.2.24 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 shrunken, 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 between two neighboring 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 alignment of the layers. There are three alignment flags that you can use for the horizontal alignment (HCenter, Left, Right) and another three for the vertical alignment (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)
```

The alignment of the layers can be done with respect to the drawing area between the axes (Graph.AlignCanvases) or with respect to the whole layer widget (Graph.AlignLayers) and you can specify the alignment policy to use via the following method:

```
g.setAlignPolicy(Graph.AlignCanvases)
```

A very often needed layout is the one with shared layer axes having linked abscissae (modifying the x scale for one layer will automatically adjust the scales for all plot layers). Here's how you can simply create such a 2x2 layers grid, with only a few lines of code:

```
g = newGraph("", 4, 2, 2)
g.setSpacing(0, 0)
g.setAlignPolicy(Graph.AlignCanvases)
g.setCommonLayerAxes()
g.arrangeLayers()
g.linkXLayerAxes(True)
```

All the examples above suppose that the layers are arranged on a grid, but of course you can add layers at any position in the plot window. In the examples below 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 existing 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)
```

7.2.25 Waterfall Plots

The waterfall graph is ideal . The graph has a pseudo-3D effect, enabling you to see variations in the Z direction. Waterfall plots are ideal for comparing variations between multiple data sets created under similar conditions. A pseudo 3D effect is generated by applying an offset to all the data curves in a 2D plot layer. You can create and customize them using the functions below:

```
g = waterfallPlot(table("Table1"), (2, 3, 4))
l = g.activeLayer()
l.setWaterfallOffset(50, 20)# x/y offsets as % of the layer drawing area width/height
l.setWaterfallSideLines(True) # draw side lines for all the data curves
l.setWaterfallFillColor(Qt.lightGray)
g.reverseWaterfallOrder() # reverse the order of the displayed curves
```

7.2.26 3D Plots

7.2.26.1 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 following 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)
```

7.2.26.2 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)
```

7.2.26.3 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()
```

7.2.26.4 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()
```

7.2.26.5 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 functions:

```
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 below:

```
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)
```

7.2.26.6 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)
```

7.2.26.7 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 predefined 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 "Miscellaneous" section.

```
g.setDataColorMap(fileName)
g.update()
```

The colors of all the other plot elements can be customized as shown below. 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()
```

7.2.26.8 Exporting

In order to export a 3D plot you need to specify a file name containing a valid file format extension:

```
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, compression = 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. The `compression` parameter can be 0 (no compression) or 1 (LZW) and is only effectif for .tif/.tiff images. It is neglected for all other raster image formats.

3D plots can be exported to any of the following vector formats: .eps, .ps, .pdf, .pgf and .svg, using the function below:

```
g.exportVector(fileName, textMode = 0, sortMode = 1, size = QSizeF(), unit = Frame.Pixel,  ↩
    fontsFactor = 1.0)
```

where `textMode` is an integer value, specifying 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.

7.2.27 Data Analysis

7.2.27.1 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 change 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 analyzes 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().

7.2.27.2 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 syntax:

```
dec = Deconvolution(table("Table1"),  "B", "C")
dec.run()

cor = Correlation(table("Table1"),  "B", "C", 10, 200)
cor.setColor("green")
cor.run()
```

7.2.27.3 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.

7.2.27.4 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()
```

You can also perform 2D fast Fourrier transforms on matrices. The FFT routine takes in this case the following parameters: *rm* - specifies the real part input matrix, *im* - specifies the imaginary part input matrix, *inverse* - specifies the direction of the FFT, *DCShift* - if this is true, the DC component will be put in the center of the result matrix, otherwise, the DC component will locate at four corners of the result matrix, *norm* - specifies whether or not to normalize the amplitudes to 1, *outputPower2Sizes* - forces output matrices whose sizes are integer powers of 2 (zero padding is used to initialize the missing cells)

```
fft = FFT(Matrix *rm, Matrix *im = NULL, bool inverse = False, bool DCShift = True, bool ↩
    norm = False, bool outputPower2Sizes = True)
```

Here's how you can perform the 2D FFT of a matrix called "Matrix1", and the inverse FFT in order to recover the original image:

```
from qti import *
m = matrix("Matrix1")
m.setViewType(Matrix.ImageView) # make sure the matrix is displayed as image

fft = FFT(m)
fft.run()
fft.amplitudesMatrix().resize(500, 400)

rMatrix = fft.realOutputMatrix()
rMatrix.hide()
iMatrix = fft.imaginaryOutputMatrix()
iMatrix.hide()

ifft = FFT(rMatrix, iMatrix, True)
ifft.run()
ifft.realOutputMatrix().hide()
ifft.imaginaryOutputMatrix().hide()
```

7.2.27.5 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`).

7.2.27.6 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, True) change data source (last ↩
    parameter enables/disables data sorting)
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 containing 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()
```

7.2.27.7 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.run()
result = integral.area()
```

The script bellow shows how to perform an integration on a data set from a table:

```
i = Integration(table("Table1"), "Table_1", "Table_2", 3, 20, True)# sorted data range from ↩
     row 3 to 20
i.enableGraphicsDisplay(False)
i.run()
result = i.area()
```

As you can see from the above examples, the numerical value of the integral can be obtained via the `area()` function.

7.2.27.8 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)
```

7.2.27.9 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()
```

7.2.28 Statistics

7.2.28.1 Descriptive Statistics

```
stats = Statistics("Table1_2")
stats.run()

print stats.mean()
print stats.variance()
print stats.standardDeviation()
print stats.standardError()
```

For all the statistic test below, once you have created a test object, you can use the following methods:

```
test.showResultsLog(False) # disable the logging of the results
test.showDescriptiveStatistics(False) # disable the display of the descriptive statistics results
test.run()

print test.statistic()
print test.pValue()
print test.logInfo()

t = test.resultTable("MyResultTable") # Returns a pointer to the table created to display the results (the table name is optional)
```

7.2.28.2 Hypothesis Testing - Student's t-Test

```
test = tTest(15.2, 0.05, "Table1_2") # One sample test, test mean = 15.2, significance level = 0.05
test.run()

test.setTestValue(15.0)
test.setSignificanceLevel(0.5)
test.setTail(tTest.Left) # or tTest.Right, or tTest.Both (default value)
test.run()

print test.t() # same as test.statistic()
print test.dof() # degrees of freedom
print test.power(0.5)
print test.power(0.5, 50) # alternative sample size = 50
print test.pValue()
print test.lcl(90) # lower confidence limit for mean
print test.ucl(90) # upper confidence limit for mean

test = tTest(15.2, 0.05, "Table1_1", "Table1_2", True) # Two sample paired test
print test.logInfo()

test = tTest(15.2, 0.05, "Table1_1", "Table1_2") # Two sample independent test
test.run()

test.setSample1("Table2_1")
test.setSample2("Table3_2", True) # Two sample paired test
test.run()
```

7.2.28.3 One Sample Test for Variance (Chi-Square Test)

```
test = ChiSquareTest(88.2, 0.05, "Table1_2") # Test variance = 88.2, significance level =  ↩
    0.05
test.run()

print test.chiSquare() # same as test.statistic()
print test.pValue()
print test.logInfo()
```

7.2.28.4 Normality Test (Shapiro - Wilk)

```
test = ShapiroWilkTest("Table3_1")
test.setSignificanceLevel(0.1)
test.run()

print test.w() # same as test.statistic()
print test.pValue()
print test.logInfo()
```

7.2.28.5 One-Way ANOVA

Please read the documentation of the TAMUANOVA library for more details.

```
test = Anova()
test.setSignificanceLevel(0.1) # default level is 0.05
test.addSample("Table1_1");
test.addSample("Table1_2");
test.addSample("Table1_3");
test.run()

print test.fStat() # F statistic = ssm/sse (same as test.statistic())
print test.pValue()
print test.ssm() # "between-group" sum of squares
print test.sse() # "within-group" sum of squares
print test.sst() # total sum of squares

test.showDescriptiveStatistics(False)
print test.logInfo()
```

7.2.28.6 Two-Way ANOVA

Please read the documentation of the TAMUANOVA library for more details.

```
test = Anova(True)
test.addSample("Table1_1", 1, 1) # Level factor A = 1, level factor B = 1
test.addSample("Table1_2", 1, 2) # Level factor A = 1, level factor B = 2
test.addSample("Table1_3", 2, 1) # Level factor A = 2, level factor B = 1
test.addSample("Table1_4", 2, 2) # Level factor A = 2, level factor B = 2
test.setAnovaTwoWayModel(2) # Fixed model = 0, Random model = 1, Mixed model = 2
test.run()

print test.fStatA() # F statistic for factor A
print test.fStatB() # F statistic for factor B
print test.fStatAB() # F statistic for the interaction
print test.pValueA() # P value for factor A
print test.pValueB() # P value for factor B
print test.pValueAB() # P value for the interaction
```

```
print test.ssa() # sum of squares for factor A
print test.ssb() # sum of squares for factor B
print test.ssab() # sum of squares for the interaction
print test.msa() # mean square value for factor A
print test.msb() # mean square value for factor B
print test.msab() # mean square value for the interaction

print test.logInfo()
```

7.2.29 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()
```

7.2.30 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 measurement 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 automate 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.

7.2.31 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 below:

```
from PyQt4 import uic

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.

7.2.32 Task automation example

Below 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 below:
#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()
```

7.2.33 Scope Changes

In recent versions the scope rules were changed to match standard Python:

- The keyword "global" refers to the module, e.g. a particular script window or column script. Outside of any function, this is the default namespace, so "global x" has no effect. Inside a function, "global x" refers to `x` in the module namespace. Previously, "global" referred to QtiPlot's global variables, and module-level variables were inaccessible from inside a function, unless marked global.
- To read and write QtiPlot's global variables, use the new special variable `globals`:

```
globals.myvar = 5
print globals.myvar
```

`globals` is shared among all modules.

If a script has any "global" declaration outside of a function, QtiPlot uses the old scope rules. Existing scripts should keep working, but for best results, update your scripts:

- If a "global" variable `x` is used only within one script, delete any "global x" that is outside of a function.
- If a "global" variable needs to be accessed outside the script that defined it, change

```
global x
x = 5
```

to

```
globals.x = 5
```

and replace all references to `x` with `globals.x`

7.2.34 QtiPlot/Python API

The complete QtiPlot/Python API can be consulted here.

7.2.35 PyQt Class Reference

The complete PyQt class reference can be consulted here.

Chapter 8

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.

Chapter 9

Index