pro Fit 6.1

User Manual

http://www.quansoft.com/

End User License Agreement

This is the license agreement between you and QuantumSoft covering your use of pro Fit and any other data, code or information included in this package (the "Software"). Be sure to read it before using the Software.

By installing pro Fit or by obtaining access to and using the Software in any other way, you agree to be bound by the tems of this agreement.

License terms:

1. The Software and its related documentation are provided "AS IS" and without warranty of any kind. QuantumSoft disclaims all other warranties, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Under no circumstances shall QuantumSoft be liable for any incidental, special, or consequential damages that result from the use or inability to use the Software or related documentation, even if QuantumSoft has been advised of the possibility of such damages. In no event shall QuantumSoft's liability exceed the license fee paid, if any.

2. Rights granted:

a) Use of the "trial" version of the Software is granted within the limits built-into the trial version application.

b) Use of the "full" version of the Software is granted for n users, where n is the number of users indicated in the purchase acknowledgement for the registration key bought from QuantumSoft or from one of its resellers. An n-user licence is deemed to be exceeded if the Software is installed on more than n computers. Exception: Right is granted within the single user license to install the Software on two computers simultaneously if both said computers are regularly used by one person only.

Within this agreement, "installed" on a computer means that the Software can be executed on said computer, either because it is stored locally on said computer or accessible by said computer through a network.

3. Competent court and law:

This Agreement shall be governed by the laws of Switzerland and the competent court is in Zürich, Switzerland.

4. If, for any reason, any provision of this Agreement, or portion thereof, is found to be unenforceable under the applicable law, that provision of the Agreement shall be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this Agreement shall continue in full force and effect.

5. QuantumSoft reserves the right to change this agreement at any time. Any changes will be announced at www.quansoft.com.

6. Pre-release software: By using any beta version (pre-release version) of the Software, you acknowledge that you are aware that such versions are for testing purposes only and that they often contain substantial errors that affect their functionality or may damage data on or functionality of a computer. Pre-release versions of the Software should never be used on computers containing data critical for your work.

QuantumSoft
Bühlstrasse 18
CH-8707 Uetikon am See
Switzerland
e-mail: profit@quansoft.com
www: http://www.quansoft.com

Customer Support: For information and customer support contact QuantumSoft at the following address:

QuantumSoft
Bühlstrasse 18
8707 Uetikon am See
Switzerland

e-mail: profit@quansoft.com
web: http://www.quansoft.com/

Table of Contents

1   Introduction

pro Fit is an interactive tool for the investigation, analysis and representation of functions and data. It is designed for users in science, research, engineering and education. The key features of pro Fit are:

- *Spreadsheet and data management*: Numerical and alpha-numerical data can be stored, transformed and analyzed in spreadsheets. Predefined and user defined algorithms can be used for data transformation.

- *Analysis of mathematical functions*: The values passed to mathematical functions and those returned from them are managed in a parameter window that allows to easily determine which *input values* must be used to calculate the *output values* of the function. The effects of any change in the inputs are previewed interactively and in real time.

- *Customized functions and algorithms*: pro Fit provides a very powerful and simple Pascal-like syntax for defining mathematical functions, data transformation algorithms, drawings, and general macros, supporting floating point numbers, complex numbers, vectors, strings, and up to 4x4 matrices as general data types to be used in any mathematical expressions.

- *Interactive parameter modeling and curve fitting*: A key feature is pro Fit's intuitive interface for modeling and fitting data, offering the choice of several fitting algorithms and optional restriction of parameter ranges. Fitting supports y- as well as x-errors and allows a Monte-Carlo error analysis for fitted parameters. A parameter can also be fitted manually by dragging the function's curve with the mouse.

- *Professional plotting*. Data and functions can be plotted accurately and flexibly. Plots can have multiple coordinate axes using linear, logarithmic, 1/x, and normal probability scalings, including reverse scaling. Plotting types include scatter plots, line plots, skyline plots, histograms, contour plots, color plots, and box plots. Interactive 3D models of function and data can be generated and edited through the 3DplotterGL plug-in.

- *Built-in drawing editor*. A complete range of drawing tools allow flexible editing and annotation of plots and presentations.

- *Customizable graphical elements*. Dash-patterns, line thicknesses, arrows, error bars, color schemes and data point symbols can all be customized.

- *Extensive graphical output possibilities*. Pro Fit supports output and, in part, input for the following graphic formats: PDF, PostScript™, EPS, PICT, high resolution bitmaps, PGF, TIF, GIF, JPEG.

- *Scriptability and Recordability*. You can record your actions automatically as a pro Fit program or Apple Script for replaying them later.

- *Externally compiled code*: Import plug-ins of functions, algorithms, and other programs written in your favorite programming language or in Apple Script.

- *On-line evaluation of mathematical expressions*: Wherever pro Fit expects numerical input (such as in spreadsheets or dialog boxes) any mathematical expression can by entered

- *Drawing from a program*: pro Fit programs can directly draw in pro Fit's drawing windows to create drawings with high precision coordinates. These drawings are available for copying and pasting into other applications and for high resolution printing. Specific drawing objects as buttons, check boxes, and pop-up menus are supported to be used as interfaces for user-defined programs.

- *Macro programming*: Write complete macros to perform common tasks such as opening and closing document windows, fitting, importing and exporting files, etc.

- *Debugging environment*: A powerful debugger provides tools for developing and debugging complex programs and functions.

- *Extensive on-line help*: An on-line help system provides answers, hints and explanations.

- *Powerful plug-ins*: Various plug-ins further increase pro Fit's power, e.g. for contour plotting and 3D plotting of functions and data sets (3D plotting requires a Power Macintosh with OpenGL).

- *And much more...*: Such customizable data file import and export, services, multi-dimensional functions, etc.

System Requirements:

pro Fit 6.1 has been developed for MacOS$^{®}$ 10.4 or better. Compatibility with earlier systems has not been tested.

For users interested in running a version of pro Fit on older systems and machines, pro Fit 5.1 through 6.0 are still available upon request:

- pro Fit 5.1 is MC 680x0 based.

- pro Fit 5.5 is Power PC based and allows to save most of its documents in pro Fit 5.1 format.

- pro Fit 5.6 requires MacOS X 10.0.4 or later, or MacOS 9 with CarbonLib 1.3 or later.

- pro Fit 6.0 requires MacOS X 10.3 or later.

A note on updates

Development of pro Fit continues. To check for updates to your current version, visit QuantumSoft's web site at http://www.quansoft.com/.

You can also use pro Fit's built-in mechanism for checking for updates. Choose "About pro Fit" from the application menu and click the button "Check for Update". pro Fit will contact our web servers and tell you about any updates that you may want to download.

How to read this manual

This manual gives a full description of pro Fit. If you do not want to read it, you will still be able to find your way through pro Fit: pro Fit has been designed to be used without a manual and most of its features are self-explanatory. Extensive on-line help is provided. However, you will need to read the manual to efficiently work with some of pro Fit's most advanced features.

If you are already familiar with pro Fit 6.0, please refer to the last section of this chapter, "Changes between versions 6.0 and 6.1". Then you may go directly to the chapters giving in-depth information on the new features. If you prefer a beginner's introduction, you should continue with the section "Getting started", which gives an overview on the most common features of pro Fit.

Basic concepts

pro Fit works with *data*, *drawings*, *functions* and *programs* and can be controlled by *AppleScripts*.
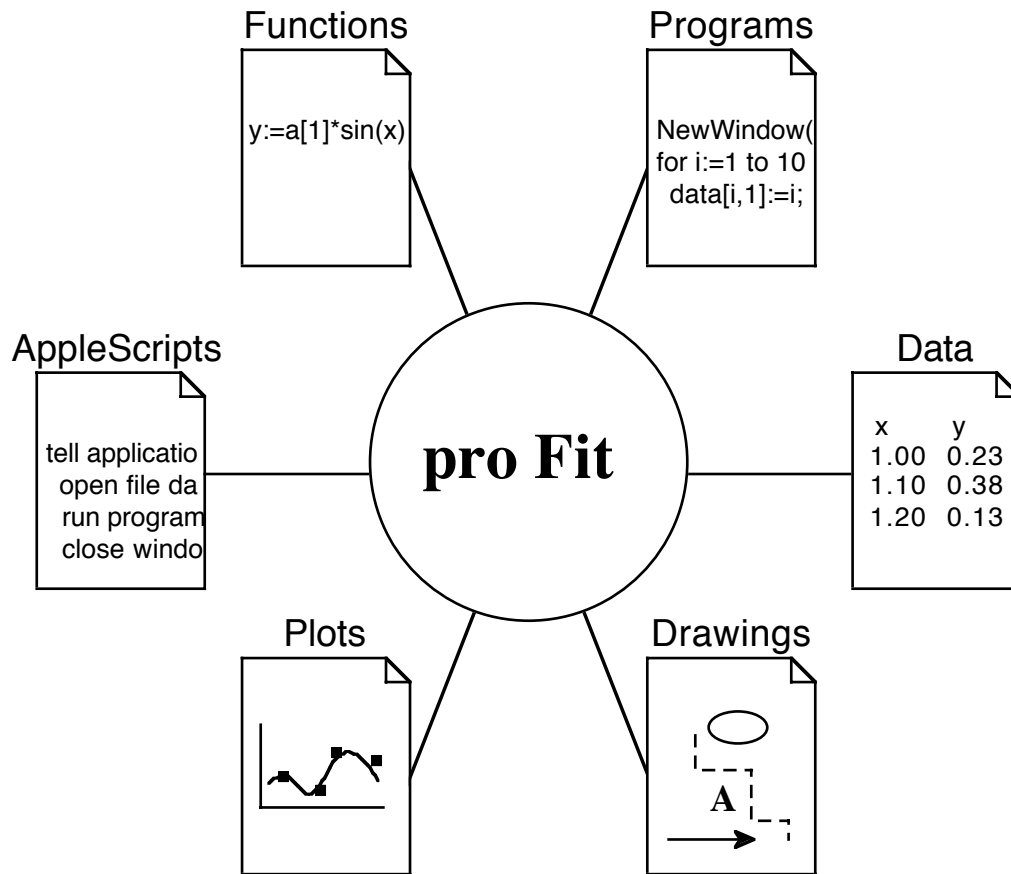
You can enter **data** into spreadsheet windows. Data can be transformed by built-in transformation algorithms, (e.g. sort, transpose, filter, Fourier transform, or mathematical operations) or by user-defined ones. Data can be text or numbers.

You can define your own data transforms by writing **programs**, which can access directly the data in the spreadsheet window. pro Fit translates these programs into computer code, which can be executed directly by the central processing unit(s) in your computer. You can **automate** many operations using such programs or Apple Events and Apple Script.

Functions can be used for plotting, analysis and fitting. There are a number of built-in functions (such as log, cos, exp, etc.). You can define your own functions using the same simple and powerful definition language used to define other programs and macros.

Functions and programs can also be defined using an external compiler (plug-ins).

You can plot your functions and data sets in a **drawing** window. pro Fit offers most standard features of a drawing program, and the appearance of all graphical elements is customizable. pro Fit generates high resolution printer information for direct printing or for exporting data via the clipboard or Drag&Drop.

Changes between versions 6.0 and 6.1

pro Fit 6.1 brings various new features. A detailed list can be found in the folder "release notes for 6.1" of the pro Fit distribution package. This is a list of the most important ones:

Fitting, analysis
- pro Fit now provides built-in support for multidimensional fitting, i.e. for fitting a function with several x- and/or y-values to a data set of the corresponding number of x- and y-columns. The command for multidimensional fitting can be found under the Calc menu.
- A new single command called Analysis combines the functionalities of the former Analysis submenu and lets you calculate the values, derivatives, integrals or roots of a function. The command is available in a non-modal window, which is updated dynamically when you e.g. change a function's parameter.

Programming
- pro Fit's built-in compiler now generates intel '386 code on intel-based Macintosh models.

Data, calculations
- Selections in data windows can now be discontinuous, not only in discontinuous rows, but in individual cells.
- The cells in a data window can now be "masked". A masked cell is invisible for all calculations.

2 Installation

The installation procedure

Installation of pro Fit is very easy. Just copy the folder containing pro Fit and is associated files to your hard disk.

Before installing pro Fit, read the "read me" file if any such file came with the package.

3 Getting started

A first session

This chapter describes a typical pro Fit session. It shows how to enter new data, plot it, and how to fit a mathematical function to it.

Our data

The world's human population is growing rapidly. Table 3.1 shows the number of inhabitants of this planet for the period after 1940

Table 3.1   The world's population since 1940

year	population in millions
1940	2200
1950	2500
1960	3000
1969	3600
1975	4000
1981	4400
1987	5000
1990	5300

Let us plot and analyze these figures.

Starting pro Fit

First install pro Fit on your computer, as described in the Chapter "Installation". Then

- **Double-click pro Fit.**

  pro Fit comes up with the following windows: The results window is used to output results of various calculations. The parameters window (titled "Polynomial" after startup in the default configuration) lists the input and return values of the current function and allows you to edit them. The preview window shows a real-time preview of the current function and data set.

  (Close these windows if you do not want them. When you need them again, choose their name from the Windows menu.)

Entering the data

First, you must enter the numbers given in Table 3.1 into a data window. To do this, you have to open a new data window.

1. Choose "New Data" from the File menu

An empty data window appears.
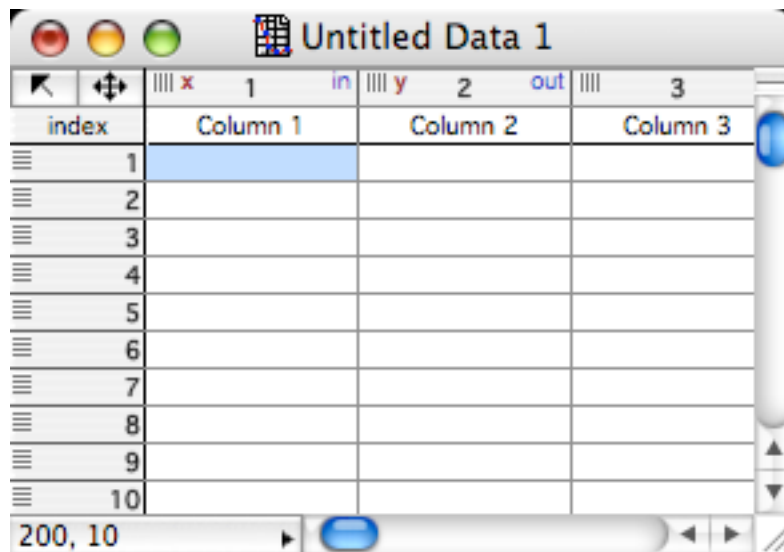
Data are arranged in horizontal rows and vertical columns. The topmost cell of each column shows the name of the column (by default 'Column 1', 'Column 2', etc.). The cells below contain the data of each column.

2. Click into the first empty cell of column 1 and enter the first year, 1940.

We fill the first column with the years and the second column with the population. The first year is 1940.

3. Click into the first cell of column 2 and enter the population in millions, 2200.

4. Repeat steps 2 and 3 to enter the other years and population figures in the following rows.

Enter the values given in Table 3.1. Note that you can use the arrow keys, the tab and the return or enter key to move from one cell to another.

5. Enter the column titles, 'year' and 'population in millions'.

Click into the titles 'Column 1' or 'Column 2' and enter the new names. Move the mouse to the vertical separation line to the right of the second column title, click, and drag the separation line a little bit to the right, so that you see the complete title. Your window should now look like this:

6. Save the data by choosing "Save As..." from the File menu.

You are prompted to enter a name for your file.

Plotting the data

Now that we have entered the data, we can display it graphically.

1. Choose "Data Y(X)..." from the Plot menu

A dialog box appears Here you can enter the ranges of the plot, the columns to be plotted, and more. In this introductory session we can use the settings as they are.

2. Click OK.

A drawing window appears, showing a graph of the data.
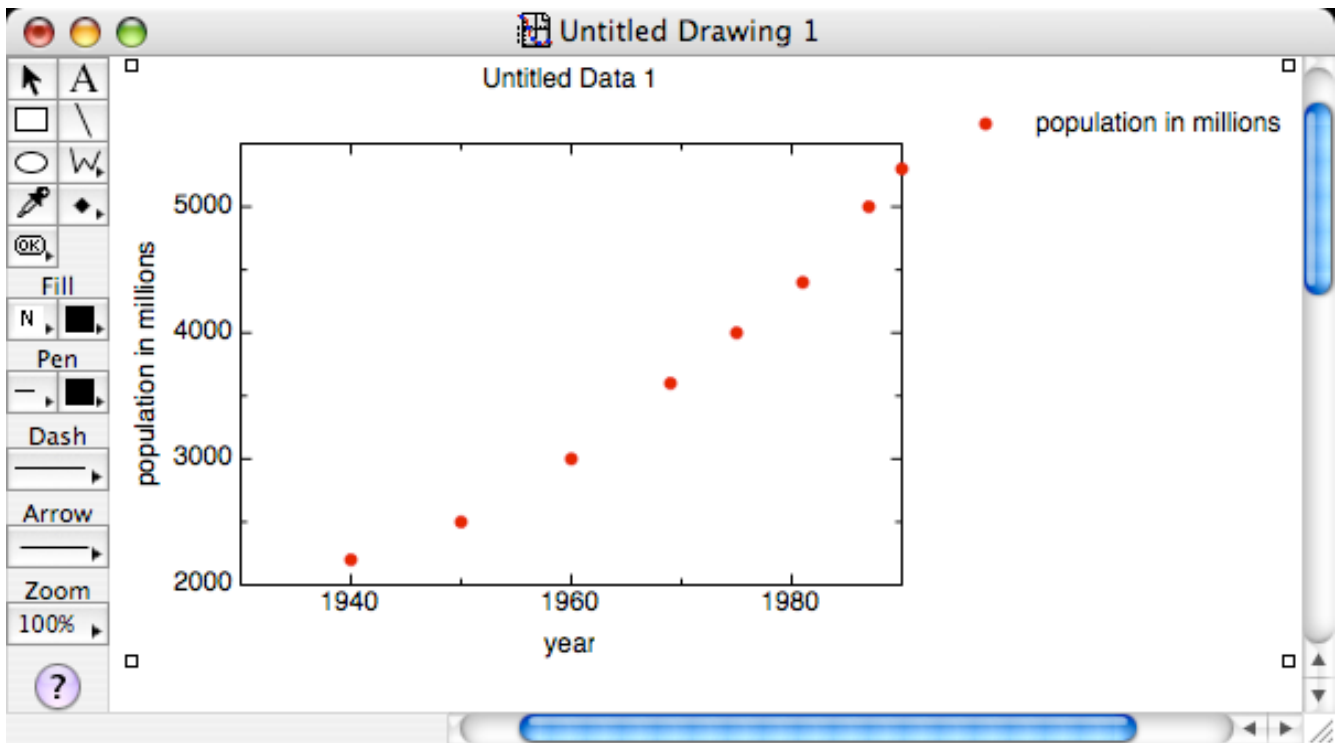
You can edit a drawing easily. For example, you can change most parts of the graph just by double-clicking.

3. Double-click the vertical axis to change its range.

(Double-click the vertical axis itself, not the numbers to the left of it!) A dialog box called "Graph Settings" appears, presenting the settings of the left y-axis:
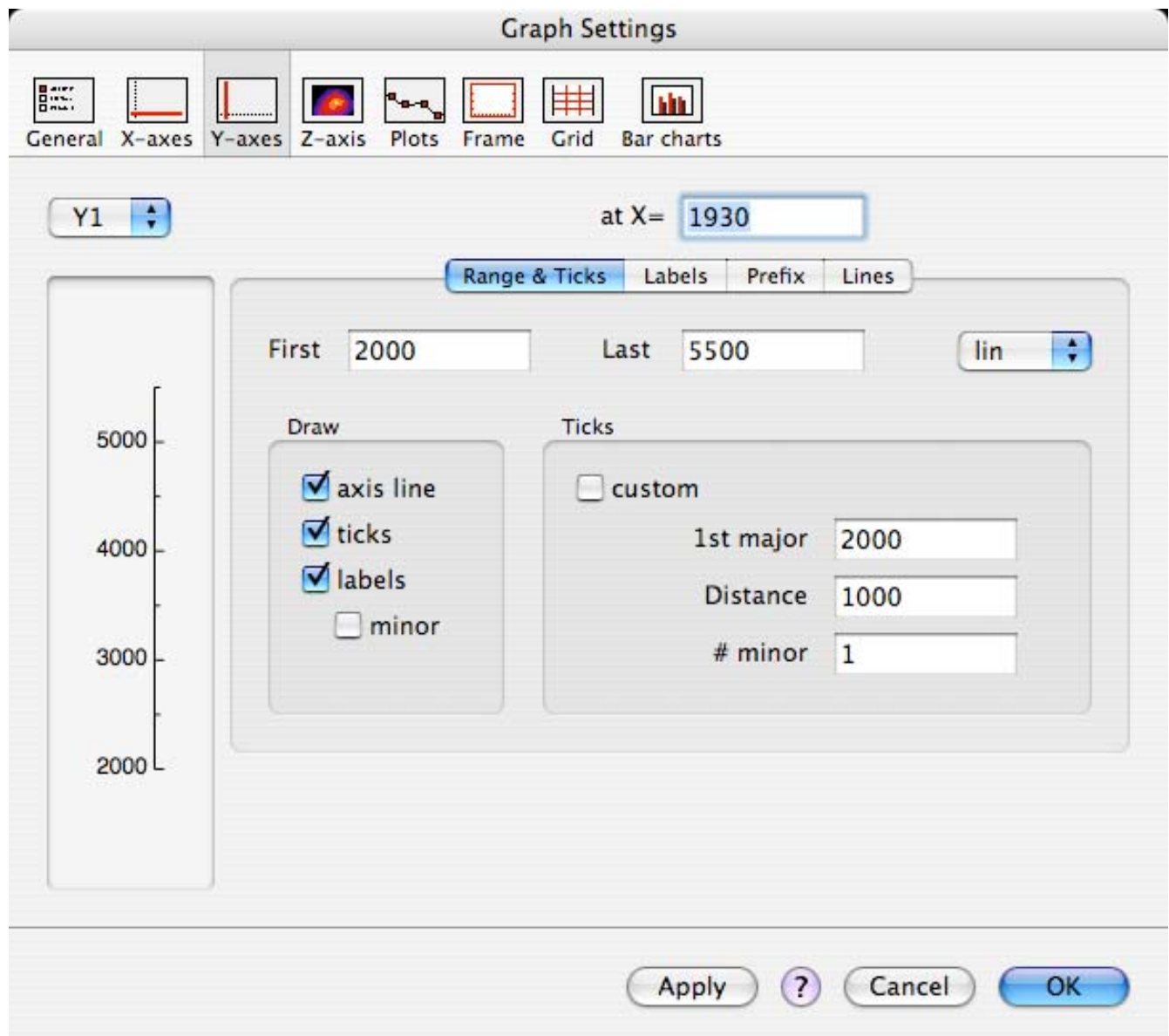
You can change a variety of parameters here. Often you will use the edit fields **First** and **Last** to set the range of the axis. Another important field is the **'Distance'** field that defines the distance between major tick marks.

4. Enter 0 for First and 6000 for Last, then click OK.

The vertical axis of the graph now starts at 0 and ends at 6000.

Double-click other parts of the graph or its legend to change other attributes. Try double-clicking the horizontal axis, the center of the plot, or the dot in the legend. You can also double-click any text in the drawing to change it. Or you can choose any of the drawing tools to add lines, polygons, text, etc.

A function to fit our data

The growth of a population can often be described by an exponential function of the type

$$p(t) = p(x_0) \times \exp\left(\frac{x - x_0}{t_0}\right),$$
(3.1)

where $p(t)$ is the population at time t, $p(x_O)$ the population at an arbitrary start time x_O, and t_O its growth constant.

Let us try to investigate the validity of this formula for the world's population. We want to find the set of parameters for which equation (3.1) fits our data best.

1. Choose Exp from the Func menu

This brings the parameters window of the Exponential function to the front. It gives a description of the built-in exponential function and its parameters:
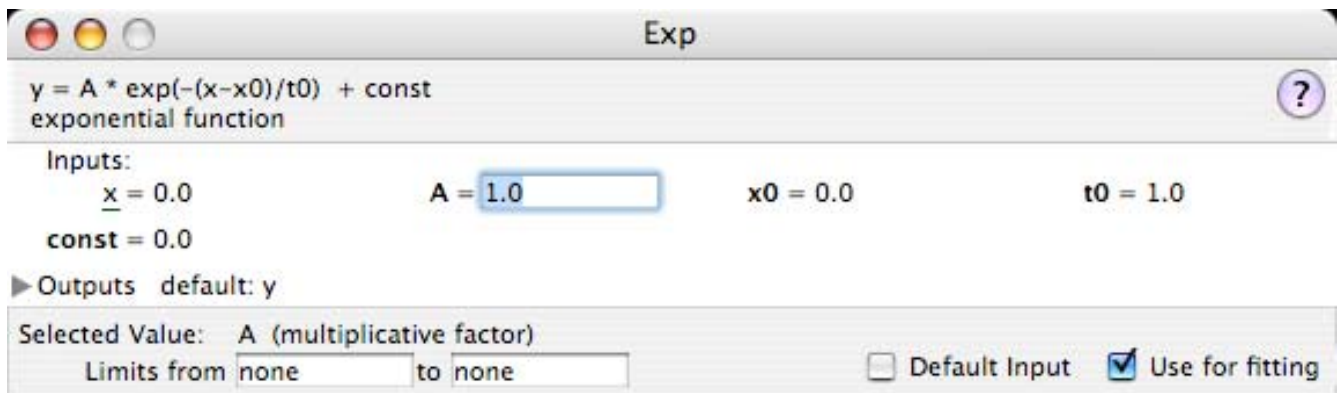
The window is divided into three regions. The top region provides a short description of the selected function and lets you load/save default parameter sets. The central part displays the input and output values of the function, and lets you edit input values. The bottom region displays additional information on the selected input or output value.

The function Exp looks like this:

$$y = A \times \exp\left(-\frac{x - x_0}{t_0}\right) + const,$$
(3.2)

which is essentially identical to equation (3.1). The parameter window also displays the default values for the parameters (input values) A, x_O, t_O and $const$. Starting from these parameters, proFit can find a better set of parameters for describing our data. But first you must define which parameters you want to fit, i. e. which parameters you want to vary in order to approximate the data with the Exponential function.

As mentioned above, the starting time x_O is arbitrary. Let us set it to 1940.

2. Click the number beside 'x0' in the parameters window and enter 1940.

This defines the parameter's value.

Since x_O is arbitrary, we do not want to fit it:

3. Uncheck "Use for fitting".

(The check box "Use for fitting" can be found in the lower right area of the window.)

The parameter name changes from bold face to plain text. This indicates that this parameter is constant and will not be fitted.

(Shortcut: You can also toggle the option "Use for fitting" by simply clicking on a parameter's name.)

4. **Click the parameter name 'const'**

We don't want to fit this parameter, either. The parameter name is not bold anymore and the option "Use for fitting" is unchecked now.

Fitting

1. **Choose Fit... from the Calc menu**

You can choose the data columns you want to fit:
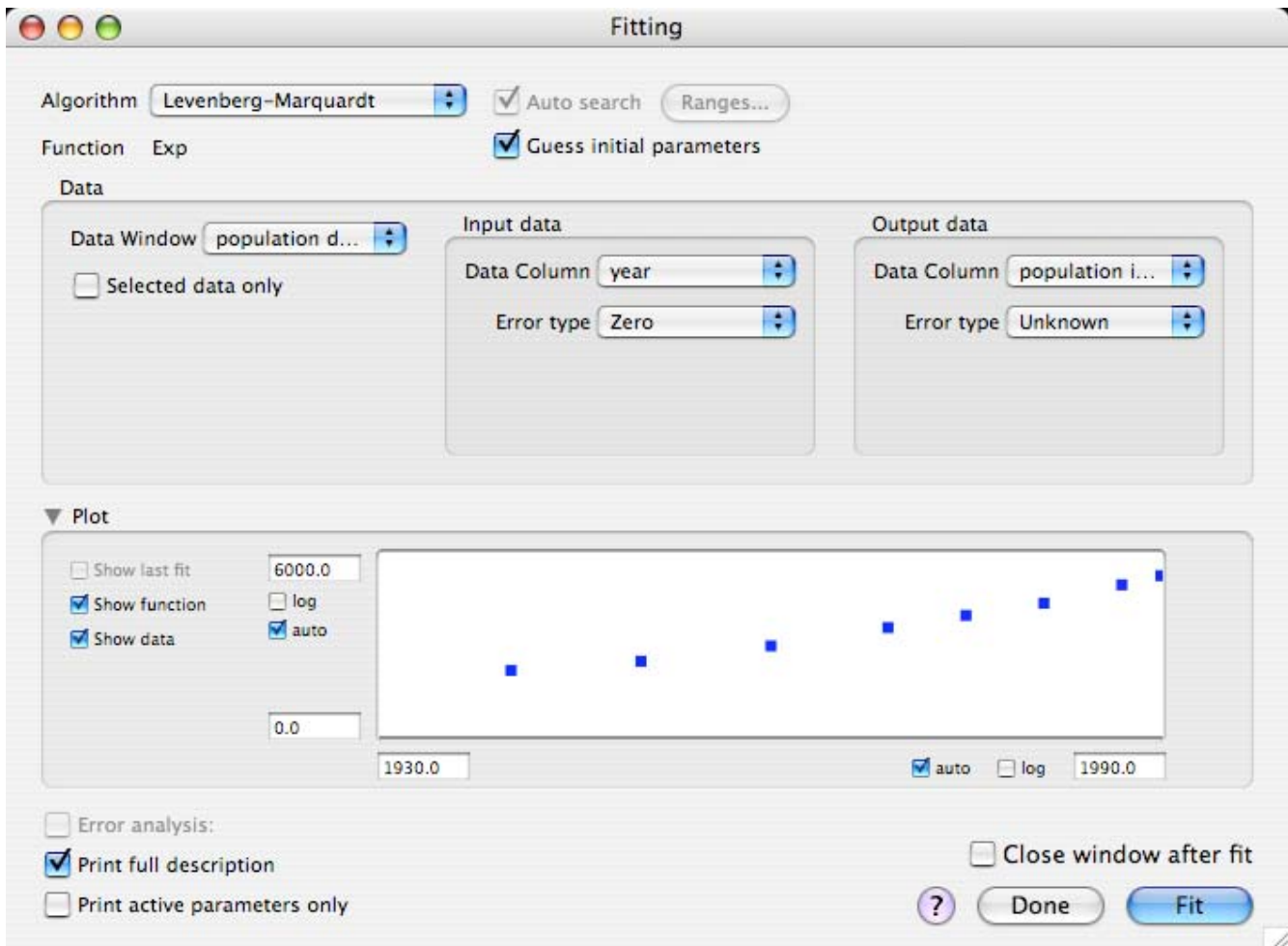
The data column settings are already ok. This box gives you also the possibility of specifying errors for the data points. For the moment, we don't need to do this.

2. **Click the button "Fit" to start fitting**

Fitting is very fast. When it is completed, the fitted function is shown in the plot preview in the lower half of the fitting window.
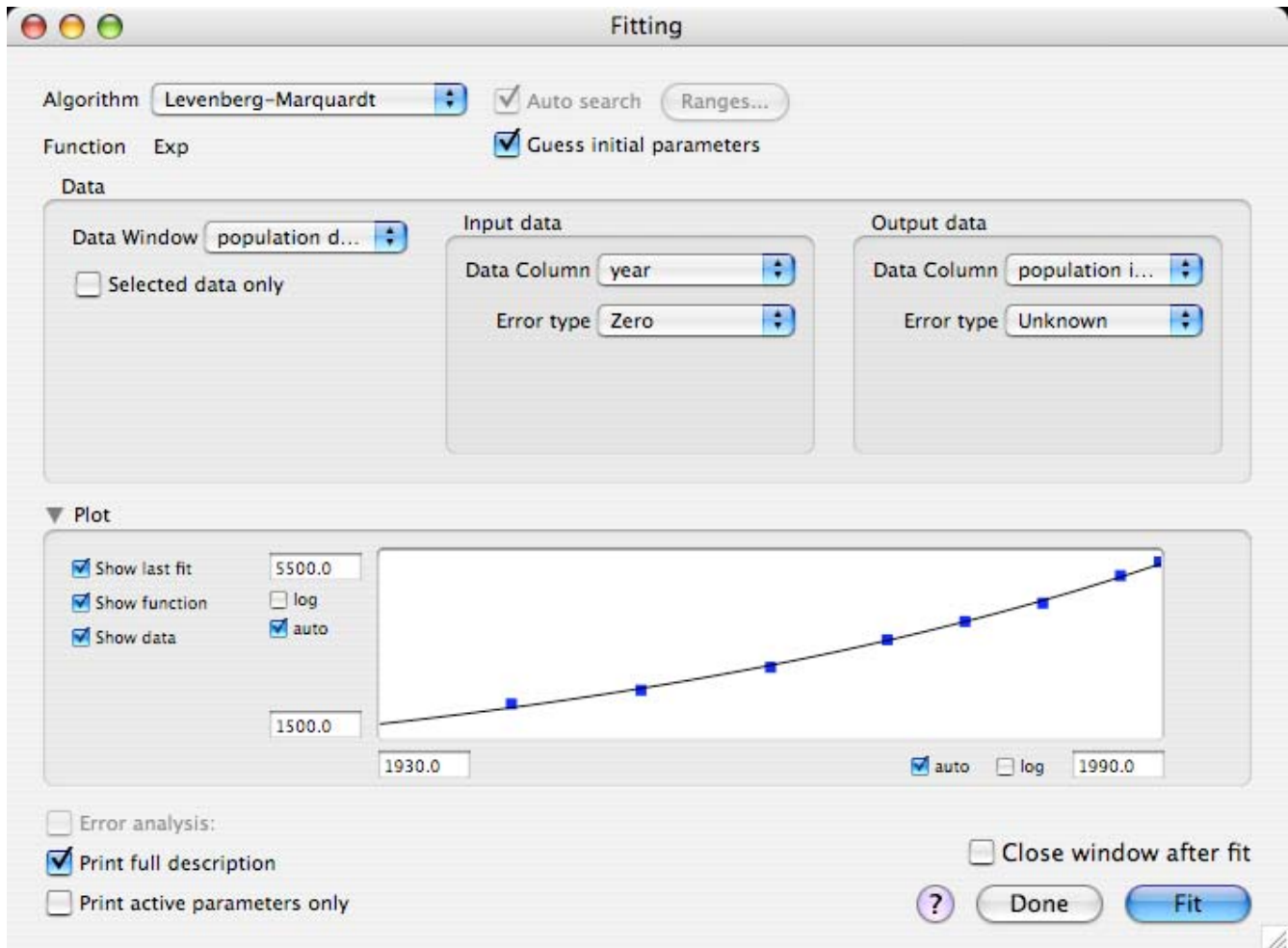
The fitted parameters appear in the results window:
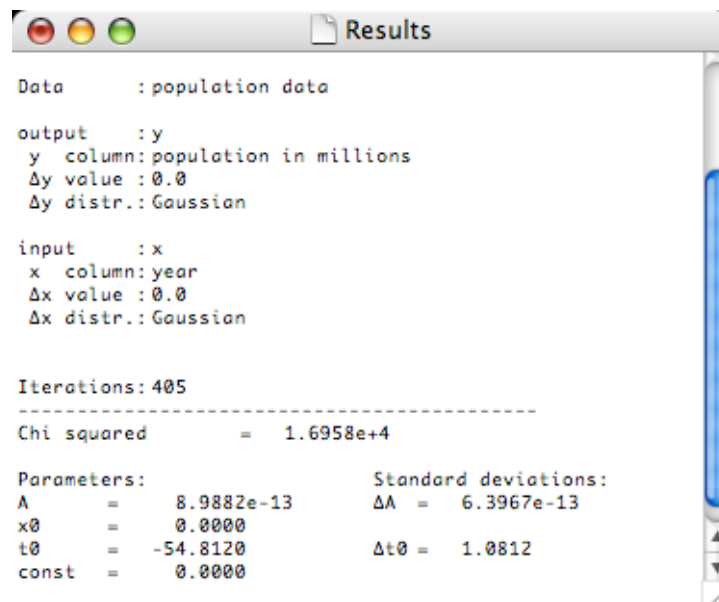
The fit yields –54.8 years for t_O and 8.99E-13 millions for A.

We can plot function (3.2) using the fitted parameters:

3. Choose Plot Function f(x)... from the Plot menu

A dialog box appears, displaying options for plotting the function:
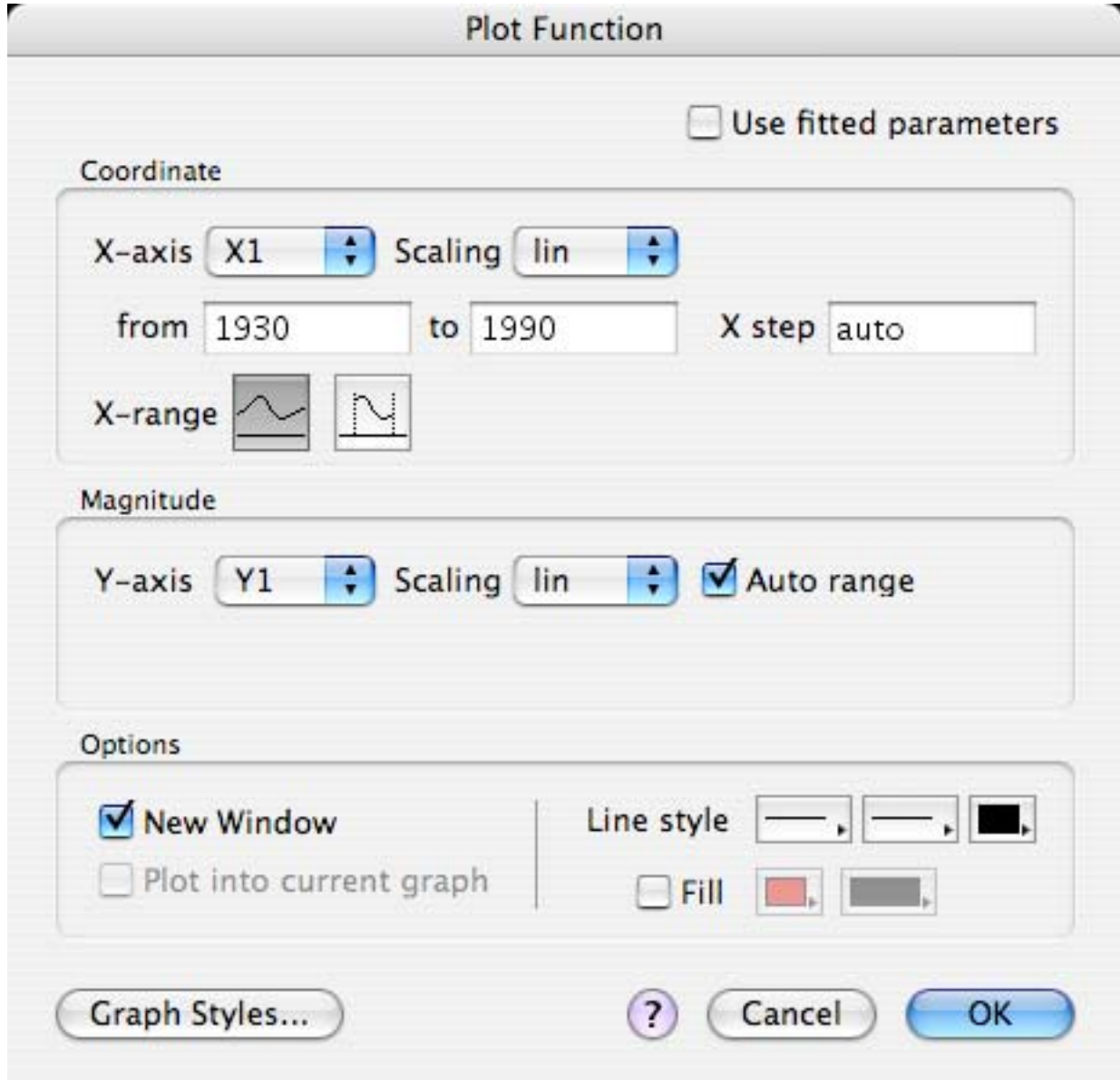
We don't need to change any of these options.

4. Click OK to draw the curve

The curve is drawn in the graph. You can now rearrange the items in the drawing window to obtain a representation of data and theory like this one:

The world's population
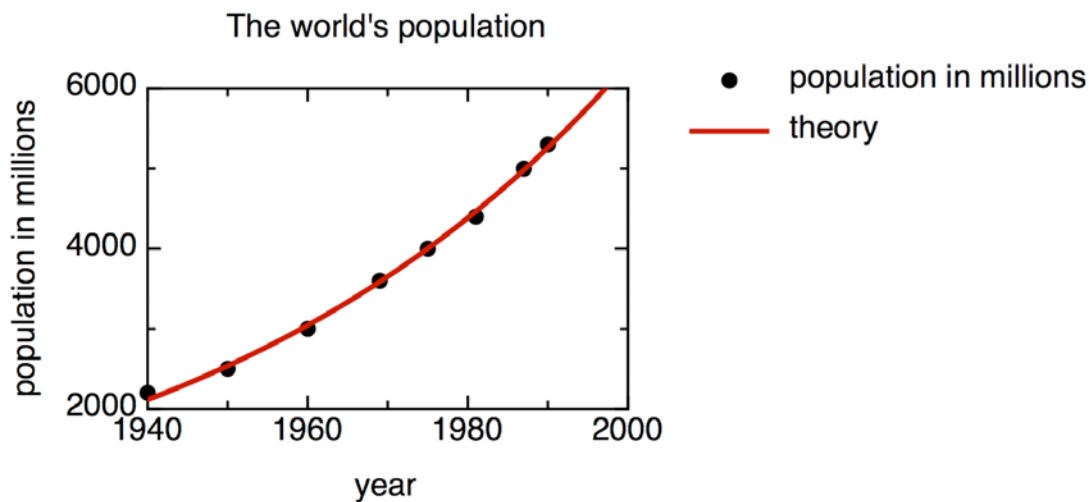
Defining your own functions

In the previous session you have fitted the built-in exponential function to your data. Fine. But what do you do if your model is described by some mathematical equation that does not appear among the built-in functions in the Func menu?

Define your own function! pro Fit can work with virtually all functions you can think of. Let us look at an example. Imagine you want to analyze a function of the form

$$y \; = \; a \sin(x) \times \ln(x) \; + b \tag{3.3}$$

with the parameters a and b. To define this function:

- **Choose New Function from the File menu.**

  This opens a new, empty function window.

- **Enter the definition of your function in the new window.**

  Just enter:

  ```
  a[1]*sin(x)*ln(x) + a[2]
  ```

  on the first line.

- **Click the "To Menu" button in the function window, or choose "Compile & Add To Menu" from the Customize menu.**

  This translates your function into computer code.

  pro Fit looks at what you wrote and sees that you used the standard input x and the standard parameter array (input values) a[1], a[2]. It therefore assumes that you want to define a new function and interprets your text accordingly.

  The new function is added to the Func menu, and the parameter window shows its default parameters.

Your simple expression is replaced by a complete, syntactically correct function definition:

```
function User_Function;
begin
    y := a[1]*sin(x)*ln(x) + a[2];
end;
```

The first line defines the name of the function as it appears in the Func menu (`User_Function` is the default proposed by pro Fit. You can change it to something like `LogSine`). Then, enclosed between `begin` and `end`, there follows the definition of the function. In the third line the function is calculated (from the variable `x` and the parameters `a[1]` and `a[2]`), and it is assigned ("`:=`") to the variable y.

Note: An alternative way to define the same function is:

```
function logSine(ampl, offset:real);
begin
    y := ampl*sin(x)*ln(x) + offset;
end;
```

In this definition, the parameters (input values) of the function are defined in the function header. The names used in the header are then used in the function body. This is the syntax used for standard PASCAL functions. pro Fit uses the parameter names defined in the function header for displaying the parameters in the parameters window.

After adding the function to pro Fit, you can change its parameters in the parameters window. You can plot the function, use it for fitting, calculations, etc.

To plot it, you should first set its parameters to reasonable values, e.g. 1 and 0.5: Enter these values in the Parameter window and choose "Plot Function..." from the Draw menu. In the dialog box that comes up, select the plotting range (e.g. the x-axis from 0 to 5). If you already have an open drawing window, you should check the option "Open New Window", otherwise your curve will be drawn into the existing graph.

Our sample function is not defined for x<=0. If you were to calculate it for a negative *x*-value, an error would occur. How–ever, the function converges to y=a[2] for x=0. You may want to expand the definition range of the function by defining $y(x) = a[2]$ for all $x \leq 0$. This can be done easily with the following modification. (click the "To Menu" button or choose "Compile & Add to Menu" from the Customize menu when you are finished.)

```
function logSine(ampl, offset: real);
begin
    if x <= 0
      then y := 0
      else y := ampl*sin(x)*ln(x);
    y := y+ offset;
end;
```

Your function could even become much more complicated than this. You can define functions that contain more than one statement, as well as variables and procedures. You can use most elements of the PASCAL programming language for defining functions.

The pro Fit package comes with more examples of function definitions. Look them up.

Writing programs

Besides defining functions for fitting and plotting, you can also define any data-generation and -transformation algorithms using the same syntax.

Let us have a quick look at a small program that fills the first column of a data window with the powers of two: 2, 4, 8, 16, etc. To define this program, again open a new function window (choose New Function from the File menu) and enter:

```
for i := 1 to nrRows do
    data[i,1] := 2 ** i;
SetColumnName(1,'Powers');
```

Then click the To Menu button. This time pro Fit recognizes that you are defining a program, not a function. It adds the program to the Prog menu and replaces your text with the syntactically correct version:

```
program User_Program;
var i:integer;
begin
  for i := 1 to nrRows do
      data[i,1] := 2 ** i;
  SetColumnName(1,'Powers');
end;
```

Note that this program starts with the keyword `program`, and not `function`. The rest of it follows the same syntax as a function definition, with the exception that no "parameters" are used.

To run the program, open a new data window and choose "User_Program" from the Prog menu. The first column of the data window will be filled with the desired values.

In this chapter, you have seen some of the most important features of pro Fit. For in-depth information consult the following chapters of this manual.

4 Working with data

Data editing

The data window

The data window is organized in horizontal rows and vertical columns. It can hold up to 16 millions columns with up to 16 millions rows if enough memory is available.
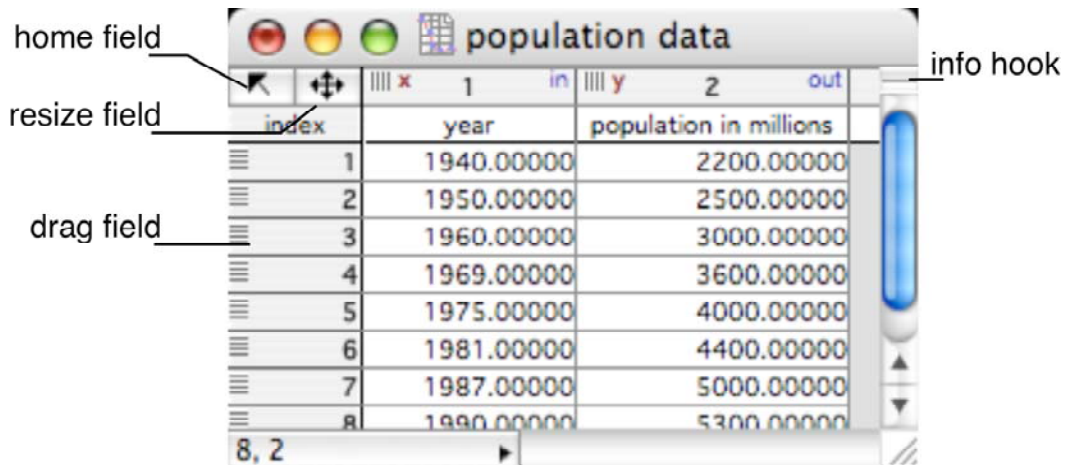
To change the size of a data window (i.e. the number of rows and columns), click the **resize field** in the top left corner of the window.

To bring the first cell of the first column into view, click the **home field** (to the right of the resize field).

To insert or delete empty rows or columns, click one of the **drag field**s and drag the mouse.

To change the width of a column, click and drag the separation line between column titles.

Dragging down the **info hook** opens an empty area at the top of the data window. In this area you can enter general information or comments about your data:
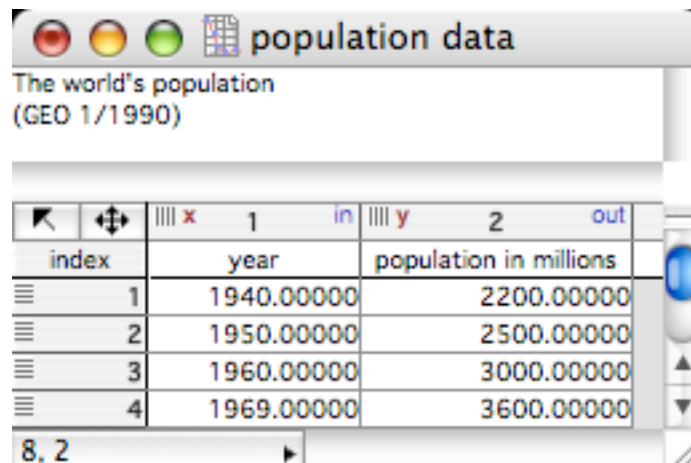
When editing numbers in a data window, the arrow keys ($\boxed{\downarrow}$ $\boxed{\uparrow}$ $\boxed{\leftarrow}$ $\boxed{\rightarrow}$) move the selection mark to neighboring data cells.

If you hold down the option key while pressing $\boxed{\leftarrow}$ or $\boxed{\rightarrow}$, the insertion mark moves horizontally within one cell.

The tab key moves the selection one column to the right. The carriage return or enter key moves the selection to the cell below.

The little "in", "out", "x", "y" symbols in the header of the first two columns tell that they will be preselected as defaults to be associated to the "inputs" of functions (the "in" columns) or to be associated with the "outputs" of functions (the "out" columns). The "x" and "y" columns are special "in" and "out" columns that are per default associated with the current default input value of a function (the "x" value) and the current default return value of a function (the "y" value). When using the simplest functions of plotting and fitting you will most likely only work with "x" and "y" columns.

Selecting data

You can select a **single cell** by **clicking** it.
- To select a **rectangular region** of data cells, drag the mouse from the top left to the bottom right cell, or click the top left cell and then click the bottom right cell while holding down the shift key.
- To select **all cells in a row/column**, click the **row/column number** field. To select several rows or columns, click and drag over the row or column numbers you wish to select.
- To select all cells in a column **starting from a certain row**, hold down the **option key** while clicking the topmost cell of the desired selection ,or click the **column number** field and then drag the mouse down to the first row to be selected.
- To select all cells, choose **Select All** from the Edit menu.

You can create a **discontinuous selection**:
- To extend or modify a current selection to a discontinuous selection, click (and drag) into the cells to be selected or deselected while holding down the **command key**.
- Note that a discontinuous selection can also be created by selecting data in the **Preview window**. See also Chapter 6, "Preview Window".

Data types

By default, each column of a data window contains numerical data, i.e. **real-valued numbers**. The precision and range of these numbers can be:
- 10^{-300} to 10^{300} with approximately 12 significant digits (double precision)
- 10^{-38} to 10^{38} with approximately 6 significant digits (single precision)

See Appendix A for details on numeric representations.

By default, a new data window opens with either single or double precision columns. The default type can be selected by choosing the command "Preferences" from the File menu. In the dialog box that comes up, click the "General" icon. See Chapter 13 for details.

A column can also contain text, up to 255 arbitrary characters in each cell. To switch between text and number formats, first select the column or columns you want to change and then choose **Column Format** from the Calc menu. Alternatively, you can also double-click the column number of a column you want to change.

After either of these actions, the Column format dialog box appears:
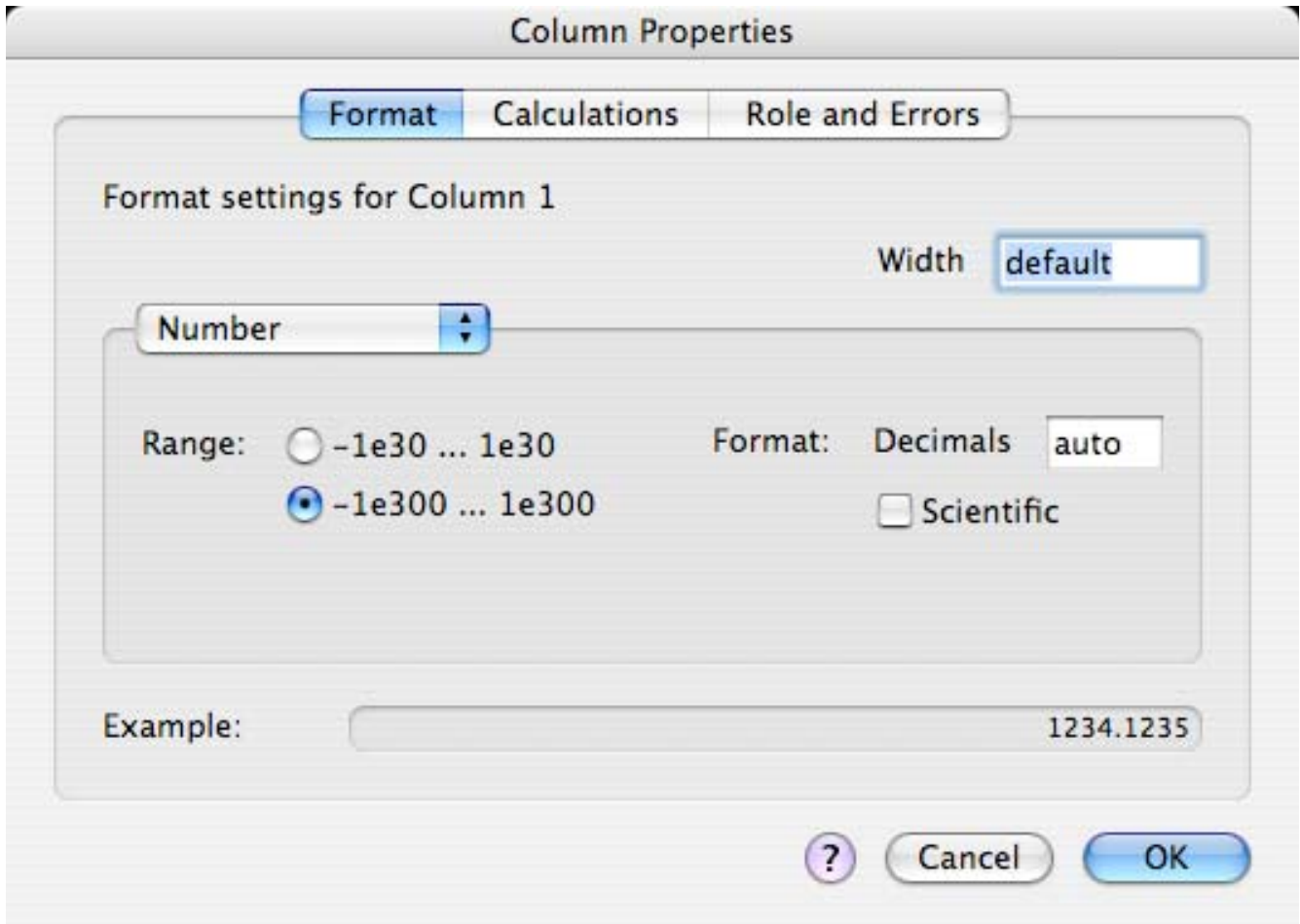
The popup at the top left contains the entries "Text", "Number", "Date & Time" and "Rel. Time". Choose **Text** if you want the selected columns to contain text, choose **Numbers** for numerical data. In the latter case you can specify their **Range** (single or double precision) and define the format for displaying numbers: select the number of digits to be displaced after the decimal point (decimals) from the **Decimals** pop-up menu. If you check **Scientific**, all numbers will be shown in exponential representation (i.e. 1.34e+3 for 1340).

You can also enter the **column width** in pixels in the corresponding edit field. A second way for changing the width of a column is to click on the boundary line between column titles (the mouse cursor will change to ⟷) and drag it to the desired position.

If you choose "**Date & Time**" as the column format, you can display a date and a time in a column.

Note: Use the "Date & Time" panel of the "Preferences..." command in the "pro Fit" menu to choose the display format for date and time values.

About dates:

The Mac OS stores dates as the number of seconds since January 1, 1904. For the technically minded, the date is stored as an integer number, 8 byte long.

pro Fit users the same convention as the Mac OS to store dates, but uses "double" floating point values instead of integers. With this number representation, pro Fit can store and recognize dates with second

precisions until up to 10^{15} (this corresponds more or less to a 6 byte long integer) seconds after January 1, 1904. .pro Fit can store dates with second-precision up to **31 million years in the future**, and it can store dates with day-of-the-week precision up to 3.1 **billion** years (3×10^{12}) in the future.

However, the date-time conversion routines currently available in Mac OS 9 only support dates up to 29'940 AD for date-to-string conversions, date-calculations, etc. Up to this limit, pro Fit can store dates with a precision of milliseconds, while it can store dates in the present with a precision of approximately a microsecond.

If you choose "**Rel. Time**" as the column format, you can display a time difference.

Note: Use the "Date & Time" panel of the "Preferences..." command in the "File" menu to define how many days fit in a year or a month.

pro Fit stores relative times as double precision floating point numbers, interpreted as a number of seconds. This leads to a range of -2147483647 to 2147483647 centuries, which corresponds to floating point numbers between -6.77680e+18 and 6.77680e+18

Permanent transformations

pro Fit data windows allow you to attach a permanent transformation to each individual column.

Whenever the data fields that are used in a permanent transformation as input data, the calculation is restarted and the results are updated. If possible, the recalculation is done only for the rows that have been changed. If you are defining permanent calculations using formulas that contain calls like 'data[,]', pro Fit always recalculates all data in the corresponding column, because the input rows are not clearly defined anymore.

Permanent calculations are created either from the Data Transform dialog (discussed below) or via the Column Format dialog (discussed above). The modification of existing permanent calculations is done via the Column Format dialog.

Note that recursive usage of column data is not allowed. E.g. if column 2 is automatically (permanently) calculated from the contents of column1, then it is forbidden to define a permanent calculation for column 1 that depends on column 2. If you are defining permanent calculations using formulas that contain calls like 'data[,]', pro Fit cannot anymore be shure that there is no recursion. Avoid the definition of such recursive calculations because they can lead to unpredicatable results!

Entering data

You can type data in the data window, copy and paste it , or drag it and drop it everywhere you want.

Instead of entering a number directly, you can enter a mathematical expression, e.g. "exp(1)" or "6+sin(π/4)", or any predefined function or variable. See Chapter 9, "Defining functions and programs" for more information about all the predefined keywords and functions you can use in mathematical expressions.

You can also import data from text files. See the Appendix C, "File Formats" for detailed information.

Data transformation

pro Fit offers various methods for transforming data: Numerical transformations, data reduction, sorting, transposing, and Fourier transforms. In addition, you can write programs that edit, manage, or create data in any conceivable way (for more information on writing such programs see Chapter 9, "Defining functions and programs").

All the commands for transforming data are found in the Calc menu and they work on the data window which is in front of all other windows.

Algebraic transformations

To make simple numerical transformations on your data, choose **Data Transform** from the Calc menu.

The transformations you can carry out with this command are of the form $y = \text{func}(x)$. You can define where the x-value comes from and where y has to be stored. You can also choose what function you want to use (note that some of these "functions" do not need an x-value).

Data Transformations

In / Out: ⦿ X: [• x–Value] ⬍ Y: [◇ Column 3] ⬍

☐ Use selected rows only (full or partial selections)

○ All selected cells   (Y cell = X cell)   Paste constant ⬍

⦿ Simple arithmetic :   Y := X [+] ⬍ [1]

○ Column arithmetic :   Y := X [+] ⬍ [• y–Value] ⬍

○ Differential/Integral : Y := [d/dx] ⬍ [• y–Value] ⬍

○ Various functions :   Y := [Polynomial(X)] ⬍

○ Formula :   Y := [x – c1*i]

☐ Make it permanent   (Apply)   (?) (Cancel)   (OK)
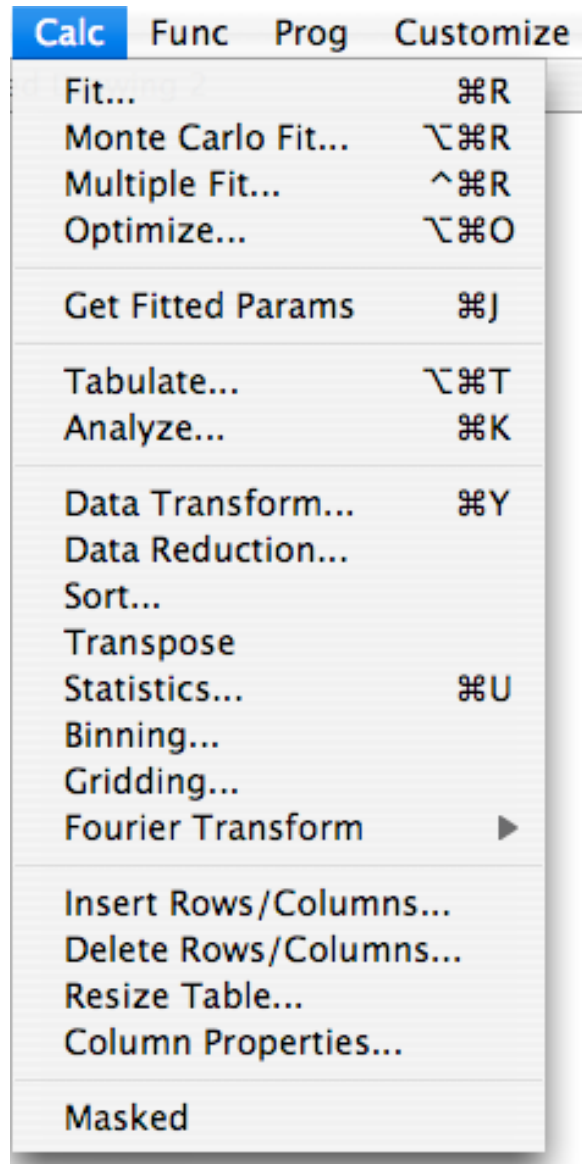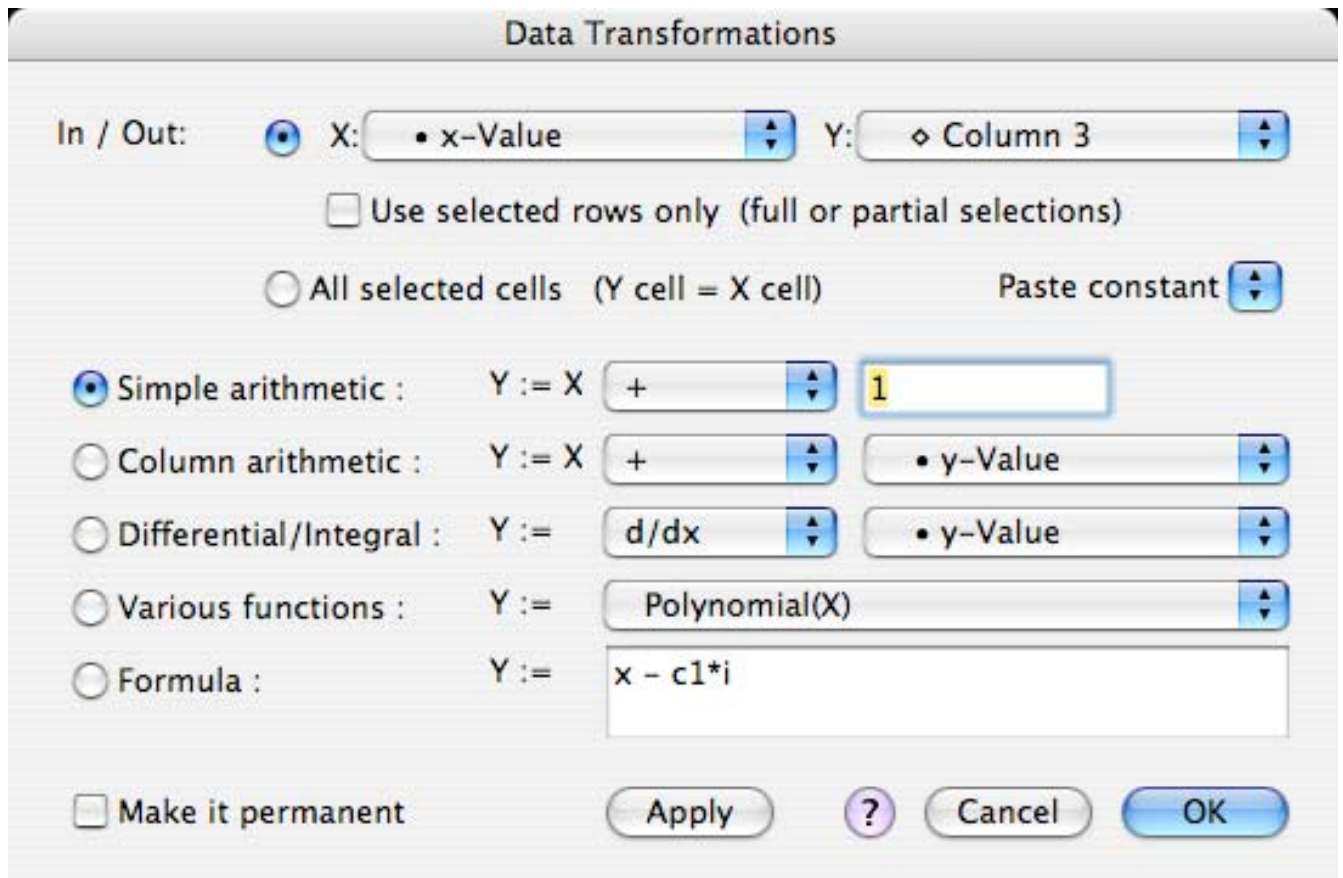
The transformation is either by columns or on the current selection: Check **Selected Rows only** to only include selected rows in the calculation. Choose **Selected cells** to work on the current selection. In calculations on the current selection, each cell (x) in the selection is replaced by its transformed value (y). In transformations by column, the cells of the x-column are transformed and stored in the cells of the y-column. You can select the x- and y-columns from the pop-up menus. (In these menus empty columns are marked with '◇', columns already containing data with '•', and text columns with '†').

Check **Make it permanent** if you want the destination column to be updated automatically whenever one of its input columns changes.

Five different groups of transformations are available:
- **Simple arithmetic**: All these transformations are of the type $y = x$ op *val*, where op is one of the operators +, -, * (multiplication), / (division), ^ (power), div (integer division), mod (modulus).
- **Column arithmetic**: These transformations are of the type $y = x$ op *col*. Again, op can be any of the operators mentioned above.
- **Differential / Integral**: These transformations return the discrete derivative or integral.
  The *derivative* is calculated as the discrete derivative of a column d that is selected from the menu to the right of the 'd/dx' popup field, in respect to the x-column. The result is stored in the y-column according to the formula

$$y_i = \frac{d_{i+1} - d_i}{x_{i+1} - x_i}.$$

The *integral* is calculated as the discrete integral of a column *d* over the *x*-column. *d* is again selected from the menu to the right of the '∫ dx' popup field. The result is stored in the *y*-column according to the formula

$$y_j = \frac{1}{2} \sum_{i=1}^{j-1} (d_{i+1} + d_i)(x_{i+1} - x_i).$$

Sometimes you may want to integrate over a *single* column *d*, or you may want to differentiate over a *single* column *d*, according to one of the following equations:

$$y_j = \sum_{i=1}^{j-1} d_i \quad \text{or} \quad y_i = d_{i+1} - d_i.$$

You can do this by creating a column containing the numbers 1, 2, 3, ... (use the fill(n) command described under 'Various functions' below) and using this column as your *x*-column.

- **Various functions**: Here you can select various simple transformation functions, such as sin(x), exp(x), ln(x), etc. Among them, you can also find the currently selected function of the Func menu, as well as the special functions fill(0), fill(1) and fill(n), which let you fill a column with the values 0 or 1 or with ascending values 1, 2, 3,... respectively.
- **Formula**: If you select this sort of transformation you are free to define any transformation statement you like. Columns are labeled by the character 'c' followed by their column number. You can use columns, constants, mathematical functions, or calls to user-defined functions in the Func menu. You can use the symbols i or n for the row number and (if you have chosen "Selection only") j or m for the column number.
  Examples:

x+sqrt(x)	an expression
tan(c10)	tangent of values in column 10
CovarMatrix(i,j)	the covariance matrix of the last fit

  The size of such a transformation statement is limited to 255 characters.

If the result of a calculation is not defined, either because a data field used for the calculation is empty or because there was an numerical error, the resulting data field is cleared.

User programs

pro Fit lets you define your own data transform programs or macros. These programs can perform data transformations in the data window, create a graph in the drawing window, etc. They are found at the end of the Misc menu.

Chapter 9, "Defining functions and programs" explains how to define such programs.

Data reduction

The command "Data Reduction" in the Calc menu offers several possibilities for data reduction, e.g. by averaging over several data points or by skipping part of the points.
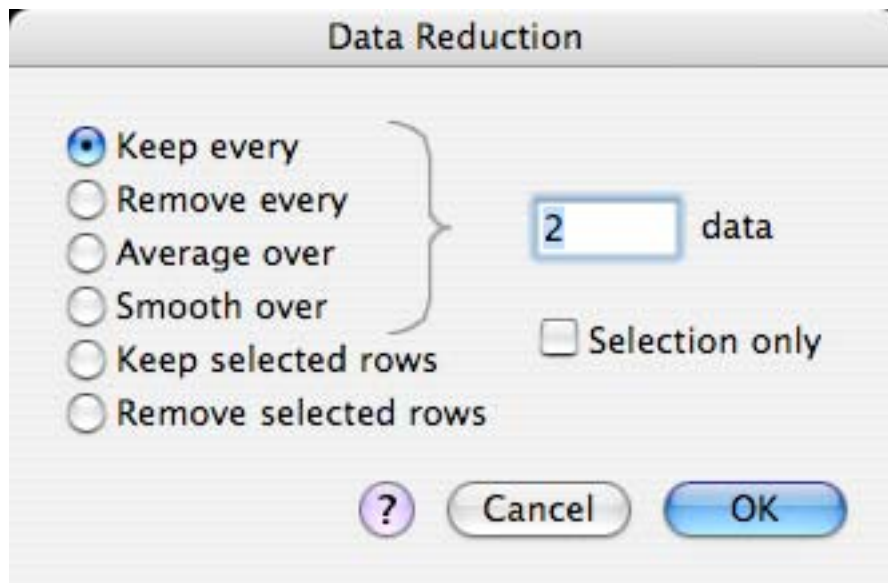
- To keep every nth row and to remove all other rows, select **Keep every**.
- To remove every nth row and to keep all other rows, select **Forget every**.
- To replace groups of n consecutive cells in a column by their average, select **Average over**. This option decreases the number of rows by a factor of n. (For example, if n=3, the values in the rows 1, 2, and 3 are averaged and the result is stored in row 1. The average of rows 4, 5, and 6 is then stored in row 2 etc.)
- To replace every data value with the average of itself and its n-1 neighboring values in its column, select **Smooth over**. Again, the average of n values is calculated. In contrast to 'Average over', the number of rows is not reduced! (For example, if n=3, the value in row i is replaced by the average over the values in rows i–1, i, and i+1).

To transform only the selected cells, check **Selection only**. In this case only the currently selected cells (highlighted in the data window) are affected.

If the selection is discontinuous (whole rows only), the above algorithms are applied to each continuous block of the selection, one after the other. The various discontinuous blocks are treated separately and do not interact with each other.

To keep only the rows that are presently selected, check **Keep selected rows**. To remove all the presently selected rows, check **Remove selected rows**.

Sorting data

To sort data, choose **Sort...** from the Calc menu:
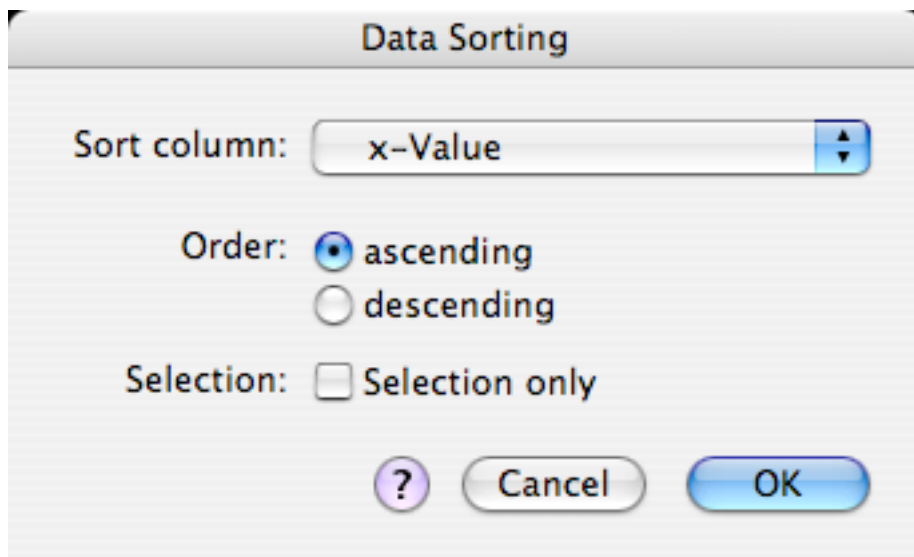
Use the pop-up menu to select the column to be used as a reference for sorting. You can sort by ascending or by descending values.

All the rows in the data window will be rearranged according to the new order in the sort column. To order only the selected part of the data window, check **Selection only**.

Note that you can only sort by columns that contain numerical data. You cannot sort by columns that contain text.

Transposing data

The command **Transpose** in the Calc menu exchanges the rows and columns in the active window. It automatically resizes the data window to make sure that all the data fits into it.

Statistical analysis of a data set

The command **Statistics...** in the Calc menu brings up a window that displays statistical data of a one-dimensional data set.

Note that this window is *non modal*, i.e. you can move it to the back by bringing any other window to the front. It always displays statistics about data in the frontmost data window.
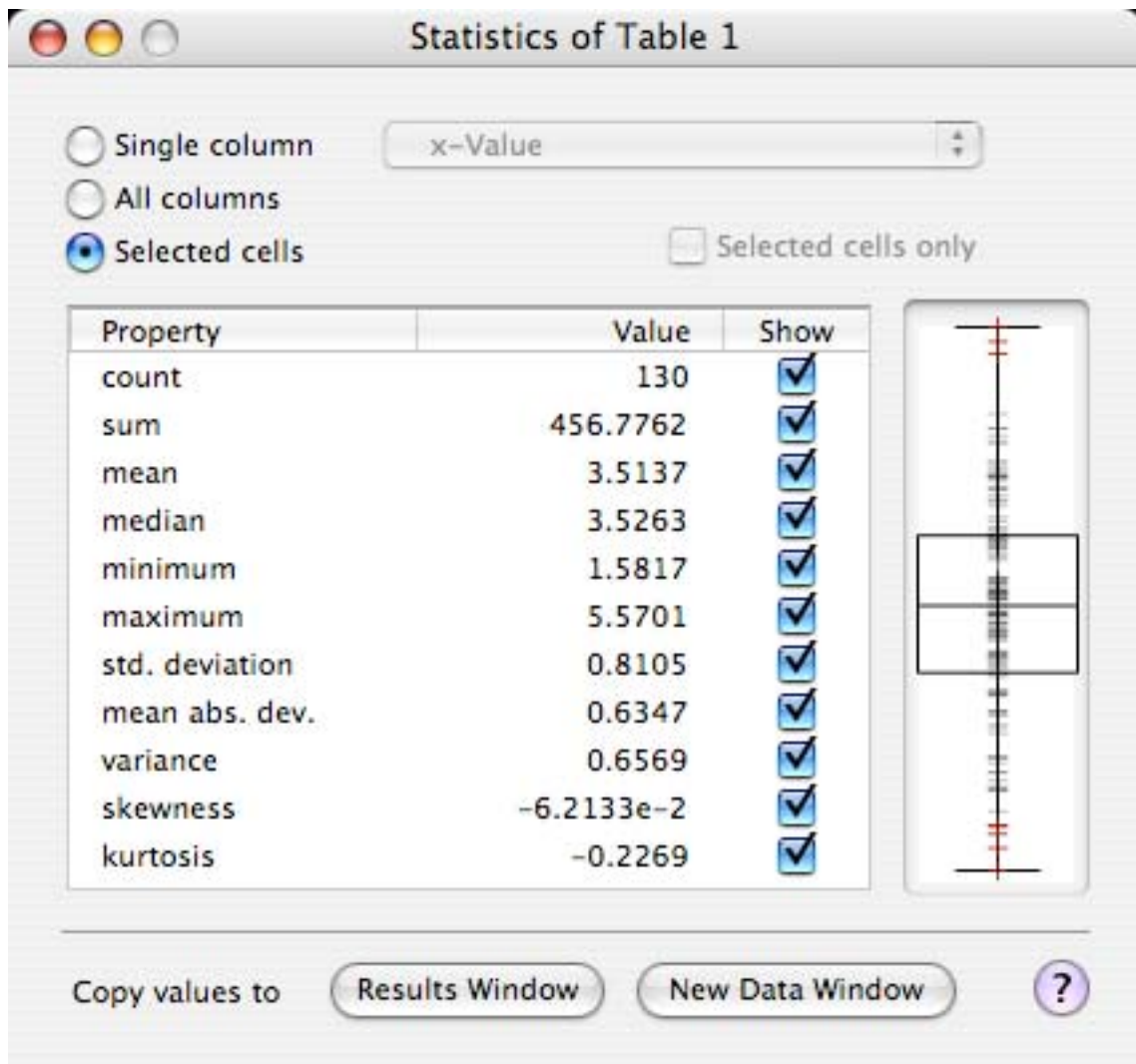
The data set that will be analyzed by the statistical algorithms can either be a **Single column** (use the popup menu to define it), **All columns**, or only the **Selected Cells**. If you specify a single column or all columns, you can check **Selected cells only** to only use the data in the selected rows.

Click **Results Window** if you want the selected statistical parameters to be printed to the results window.

Click **New Data Window** if you want the selected statistical parameters to be displayed in a new data window. This option is particularly useful if you analyze data in several columns – in that case, the statistical parameters of the individual columns are stored in separate rows of the resulting data window.

The boxplot symbol to the right of the dialog box shows the median, lower and upper quartile and the data points in the data set. Close and far outliers are drawn in color.

The following statistical values are calculated from a set of data $x_1 .. x_N$ and are printed to the results window and/or stored in a data window:

• The number of valid values N in the data set.

- The median of the sorted data set (central value for odd N or average of the two central values for even N)

- The minimum (smallest value) and the maximum (largest value)

- The sum of all valid values $\quad S = \displaystyle\sum_{i=1}^{N} x_i$

- The mean $\quad \bar{x} = \dfrac{S}{N} = \dfrac{1}{N}\displaystyle\sum_{i=1}^{N} x_i$

- The variance $\quad Var = \dfrac{1}{N-1}\displaystyle\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2$

- The standard deviation $\quad \sigma = \sqrt{Var}$

- The mean absolute deviation $\quad ADev = \dfrac{1}{N}\displaystyle\sum_{i=1}^{N}\left|x_i - \bar{x}\right|$

- The skewness $\quad Skew = \dfrac{1}{N}\displaystyle\sum_{i=1}^{N}\left[\dfrac{x_i - \bar{x}}{\sigma}\right]^3$

- The kurtosis $\quad Kurt = \left\{\dfrac{1}{N}\displaystyle\sum_{i=1}^{N}\left[\dfrac{x_i - \bar{x}}{\sigma}\right]^4\right\} - 3$

The **Skewness** characterizes the degree of asymmetry of a distribution around its mean. The **kurtosis** measures the relative "peakyness" or flatness of a distribution.

Binning

Binning is the process of putting data into bins. You define a data range that encloses all your data values. This data range is divided into consecutive intervals, the "bins". For each bin, the number of data values that lie in its interval are counted.

Example

You analyze the height of 1000 people. You put all height values into a data window. Now you want to plot a histogram, each histogram giving the number of people that have a height in a given 2 cm interval. For this purpose, you choose "Binning" from the Calc menu, choose your data, define bins of 2 cm width, and run the command.

The binning command brings up the following dialog box:
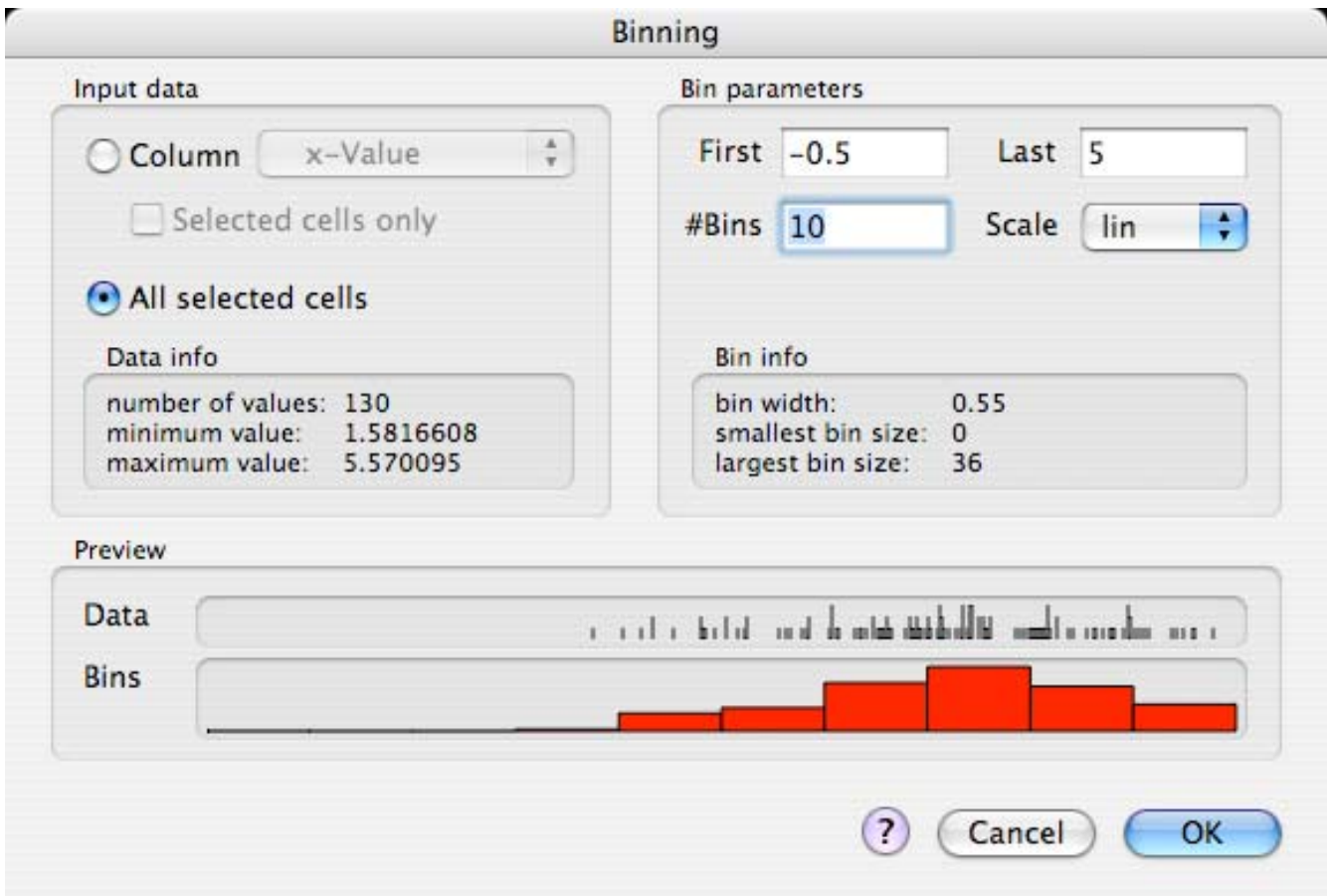
Under **Input data**, you specify the data to be binned. **Bin parameters** defines the location and distribution of the bins. The bins can be distributed linearly, logarithmically, or according to any other scaling type supported by pro Fit. For example, if you want equidistant bins, use linear scaling, if you want to have one bin for each decade, use logarithmic scaling, etc.

Under **Preview**, the position of the input data values as well as the resulting bins are displayed.

When you click OK, the center and size of each bin is calculated and the results are displayed in a new data window.

Gridding

Imagine you have a series of x- and y- values xi and yi and a z-value zi for each pair of values xi, yi. The coordinates xi, yi are not necessarily on a grid. You now want to interpolate the z-values and calculate them at a number of x- and y-values lying on a rectangular grid. This interpolation process is called gridding.

To grid a data set, choose „Grid Data..." from the Calc menu.
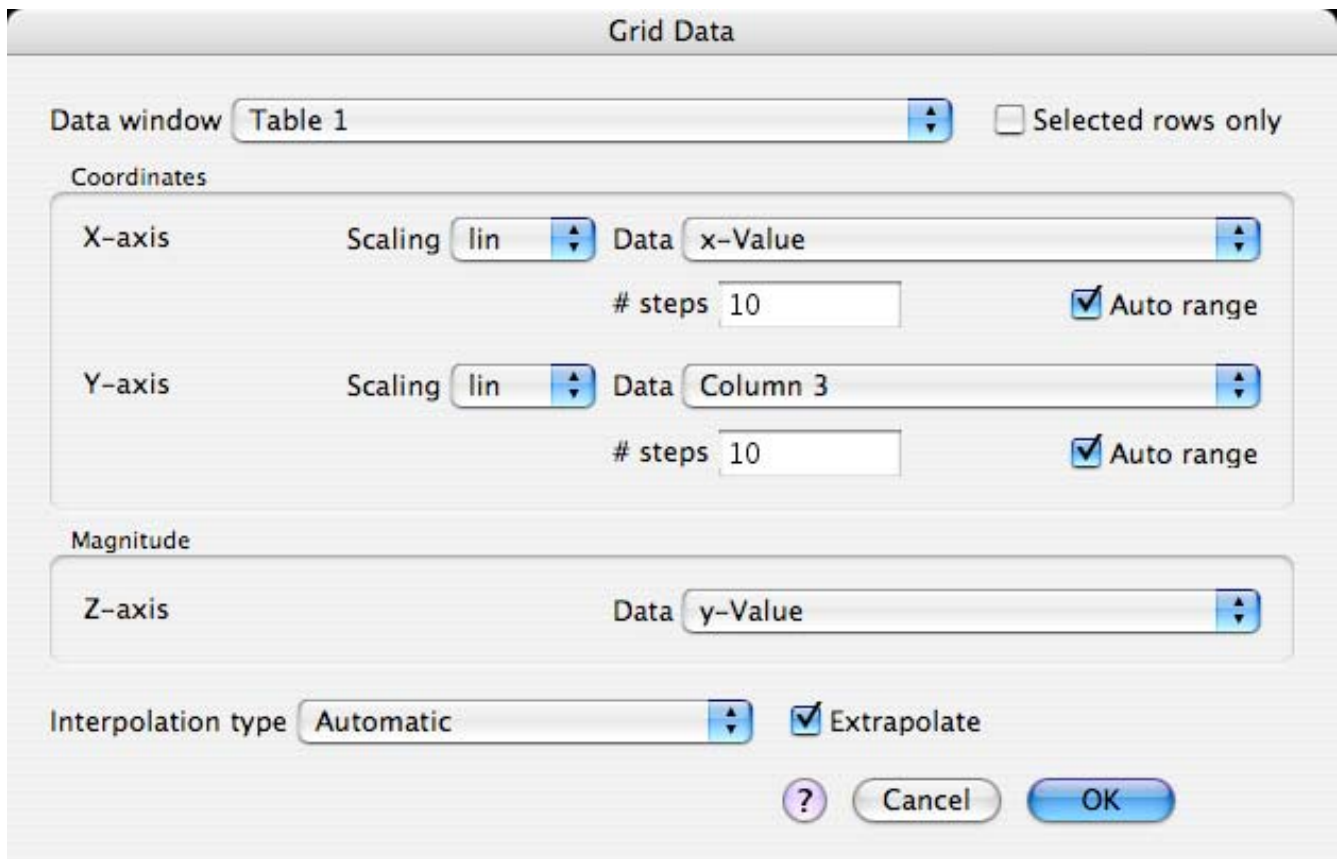
In the section **Coordinates** you can identify the X and Y-columns to be used, as well as the number of steps the grid should have in these coordinates. In the section **Magnitude** you can select the z-column to be used.

The **Interpolation type** defines the interpolation algorithm to be used for calculating z between the x- and y-coordinates of the data points. pro Fit supports the following interpolation types:

- **Automatic**: This algorithm uses the "nearest neighbor" algorithm. If that fails, it uses the robust "averaging" algorithm.

- **Nearest neighbor** (also called Natural Neighbors): This algorithm triangulates the space of x- and y-coordinates, i.e. it creates a field of triangles that covers the convex hull of the points in the x- and y-plane. When asked to return the z-value for a given x/y-coordinate, it first determines the triangle that encloses this coordinate, and then returns the z-value from a linear interpolation of the z-values at the corner points of the triangles. The code we use was derived from work by Pavel Sakov and is used with his kind permission. The algorithm requires at least 3 data points that are non-degenerate and non-collinear in the x-y-plane.

- **Modified Shepard**: This algorithm calculates a local least squares fit of a quadratic surface around each x/y/z data point. It requires at least 9 data points that are non-degenerate and non-collinear in the x-y-plane.

- **Quadratic**: Similar to the nearest neighbor algorithm, this algorithm triangulates the space of x- and y-coordinates and carries out a local interpolation in each triangle. The interpolation,

however, is quadratic. The algorithm requires at least 3 data points that are non-degenerate and non-collinear in the x-y-plane.

- **HZDS**: The Haber-Zeilfelder-Davydov-Seidel algorithm uses a technique described by J. Haber, F. Zeilfelder, O.Davydov, H.P. Seidel in "Smooth approximation and rendering of large scattered data sets", Proc. IEEE Visualization 2001, see http://www.math.uni-mannheim.de/~lsmath4/paper/VIS2001.pdf.gz. It requires at leat 10 data points that are non-degenerate and non-collinear in the x-y-plane.

- **Averaging**: Averaging is a crude interpolation algorithm based on an weighted average of the z-values of all data points, with the weight defined by the square of the distance between the desired coordinate and the each data point's coordinate in the x-y-plane. The major advantage of this algorithm is that it works well even if you only have one data point.

The result is stored in a new data window.

Fourier transforms

pro Fit can calculate Fourier transforms of numerical data. A Fourier transform is a transformation of numerical data from the "time domain" into the "frequency domain", or vice versa.

If you have a one-dimensional set of real valued data points, h_k ($k = 0 .. N–1$), the *discrete Fourier transform H_n* of these points is given by

$$H_n = \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N} ,$$ (1)

where *n* goes from $–N/2$ to $N/2$ (*N* is assumed to be even).The inverse Fourier transform is the inverse operation: It allows the calculation of data h_k in the time domain from data H_k in the frequency domain by

$$h_k = \frac{1}{N} \sum_{n=0}^{N-1} H_n e^{-2\pi i k n / N} .$$ (2)

Note that h_k as well as H_k can be complex values.

A classical interpretation of the Fourier transformation is the following:

A signal $h(t)$ is sampled at a regular time interval Δt, resulting in a set of data points $h_k = h(\Delta t\ k)$. The Fourier transform of this set of data corresponds to the frequency spectrum of the signal. H_n corresponds to the amplitude of the signal at frequency $n/(N\ \Delta t)$.

Note that the maximum frequency of the frequency domain is $f_c = 1/(2\Delta t)$, the *Nyquist critical frequency*.

The Fourier transform and its inverse can be calculated with two commands in the submenu Fourier Transform of the Calc menu: **FFT** and **Inverse FFT**. ("FFT" stands for "Fast Fourier Transform", an efficient algorithm for the calculation of the Fourier transform.)

These built-in algorithms assume that the data set h_k of the time domain is real-valued and not complex. In this case the frequency domain data set H_n is complex but we have $H_n = H_{-n}^*$ i.e. the values at positive frequency are the complex conjugate values at negative frequency. It is therefore sufficient to calculate only the positive frequency spectrum of the Fourier transform.

Note: The built in Fourier transform works on real valued data in the time domain. To work in complex data, use the plug-in "FFT" that comes with pro Fit.

To carry out a Fourier transformation, bring the data window with your data in the time domain to the front and choose **FFT** from the Fourier Transform submenu:
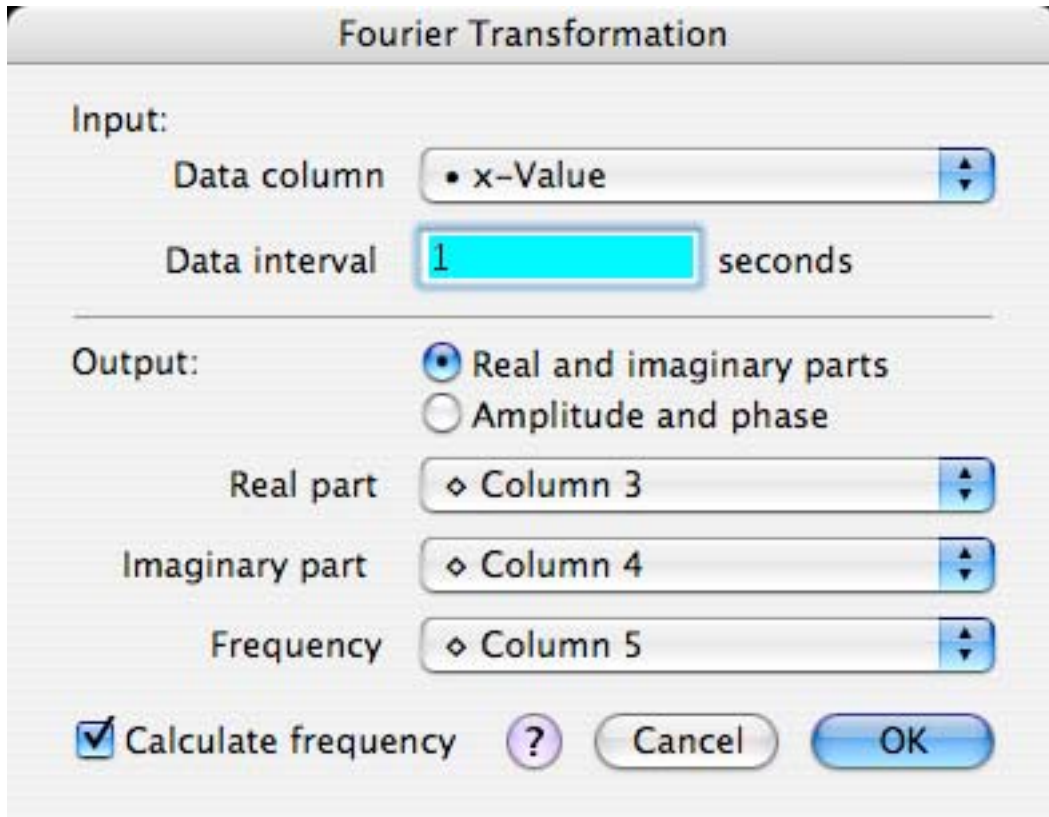
Select the column that contains your time domain data and the columns for the real and imaginary parts of your frequency domain data. If you check the box **Calculate frequency**, you must enter the time interval between two points of the time domain (**Data interval**) and a column (**Frequency**) for the frequency values of the frequency domain data.

Instead of calculating the real and imaginary parts in the frequency domain, you can also calculate their absolute value and complex argument (check **Amplitude and phase**, instead of **Real and imaginary parts**).

To calculate the inverse Fourier transform, select **Inverse FFT** from the Fourier transform submenu. The dialog box that appears for this command is very similar to the dialog box we have just seen:
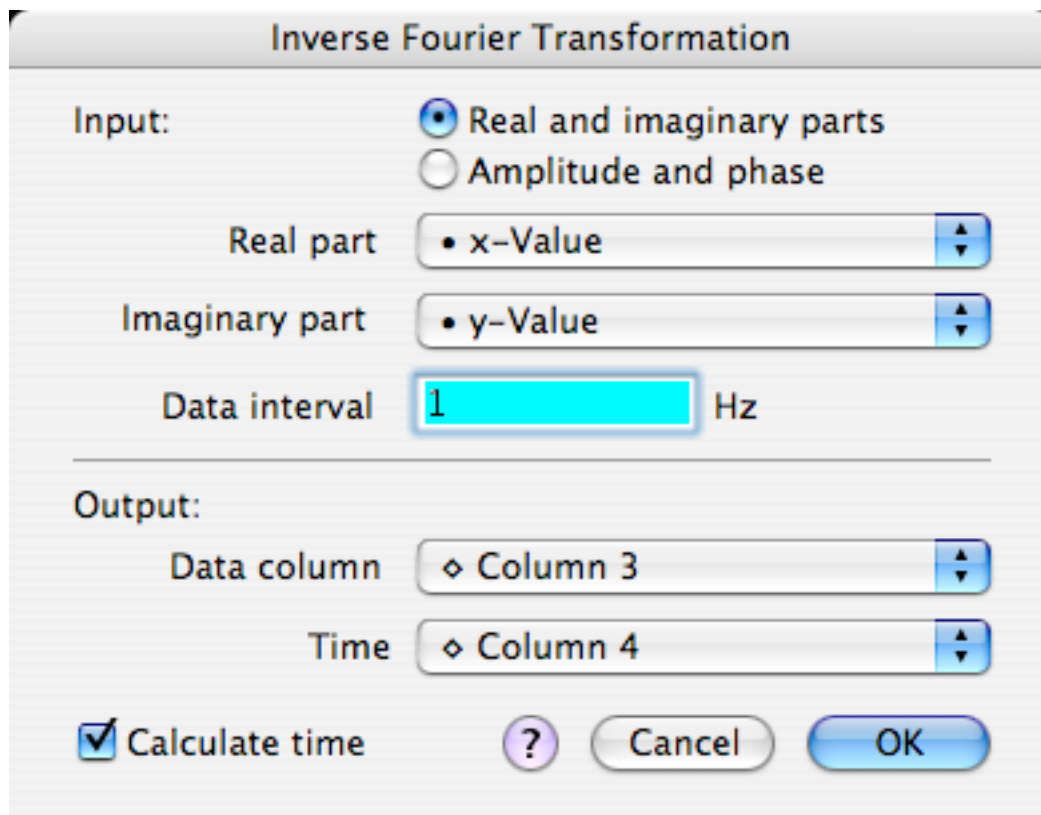
Your input data are the complex values in the frequency domain. You select the columns for the real and the imaginary parts, or, alternatively (when you select **Amplitude and phase** instead of **Real and imaginary parts**), you select the columns for the absolute value and the complex argument of your data.

The output column contains the real valued data points of the time domain.

If you want to calculate the time range (in seconds) for your output data, check **Calculate time**, enter the frequency interval between consecutive data points of the frequency domain, and select a column for the time values.

Note that if you have N points in the time domain, you obtain $N/2+1$ (complex) points in the frequency domain and vice versa.

The FFT algorithm works only for $N = 2^m$ (where m is a positive integer), i.e. for $N = 2, 4, 8, 16, ...$ If the number of input data points is not a power of two, then the missing values to the next power of two are assumed to be 0.

For further information on the subject of discrete Fourier transformations see e.g. W.H Press et al., *Numerical Recipes*, Cambridge University Press (Cambridge, 1989).

Inserting and deleting columns and rows

The Calc menu provides three commands for inserting and deleting rows and columns of a data window.:

- **Insert Rows/Columns...**: This command inserts an arbitrary number of rows or columns. The position where the rows/columns are inserted is given by the current selection. You can either insert them after the current selection or at the beginning of the current selection.

- **Delete Rows/Columns...**: This command deletes the currently selected rows or columns.

- **Resize Table...**: This command allows you to resize a data window, i.e. to change ist number of rows and columns. The rows/columns are inserted/deleted at the bottom and left of the data window. As an alternative to choosing Resize Table from the Calc menu, you can click the resize field at the top left of the data winodw.

Defining a data set to work on

Some of pro Fit's commands access data in the data windows. If you have several data windows open at the same time, pro Fit uses some rules for selecting the data window it works on:

- The transformation commands in the Calc menu work on the active data window (the window in front of all other windows). If the active window is not a data window, these commands cannot be used. (Example: the Data Transformation... command is only enabled when the front most window is a data window.)

- Some other commands use the front most of all data windows. It does not matter if windows of other kinds are in front. If the active window is not a data window, these commands look at all the windows behind the active window and work on the first data window they find. This will be the data window that is closest to the front. (Example: the Spline function uses the data window that is closest to the front.)

- The commands for curve fitting and for plotting data display dialog boxes where you can choose the data window from a pop-up menu. (Examples: Plot Data... and Nonlinear Fit...)

The data window containing the data used in a particular operation is called *the current data window* in this manual. The current data window is either the foremost data window or the window you have selected yourself.

When a data window is used as the current data window by a function or by some commands, four of its columns can have a special meaning. They are the **input and output columns**. You can define these columns using the pop-up menu that appears when you click the column number of a data window while holding down the control or command key (click on the "index" header of the index column to define the index column as one of the default columns).

For example, the 'Spline' function uses the data in the default input and output columns of the foremost of all the data windows (other windows of a different kind, e.g. a drawing window, can be active).

A small 'in', 'out' appears in the head of a column marks the default columns.

Note that the 'index' column can also become one of the default columns.

5   Working with functions

Functions in pro Fit take a set of numerical values (*input values*) and convert them into one or more *output values* by applying a mathematical transformation.

All the possible inputs of a function can be divided into a standard *x-value*, and a number of (optional) parameters that have their own individual names. The *x-value* is normally the *default input value* of the function (the quantity that is varied on the horizontal axis when plotting the function), but it is possible to also set one of the parameters to be the default input value.

A function can also have a set of *outputs* with their own individual names. Many functions have only one output, and in such a case its name is very often *y*.

Functions can be plotted, used for curve-fitting, and analyzed in various ways. The functions that are available in pro Fit are listed in the Func menu, and the function that is currently in use (the *current function*) is checked. The inputs and outputs of the current function are always visible in a dedicated window that carries the name of the current Function. We call this window the *Parameters Window*. It can be closed, but the name *Parameters* always appears in the Window menu and can be selected to make the parameters window visible or bring it to the front. Here is the parameters window of the polynomial function
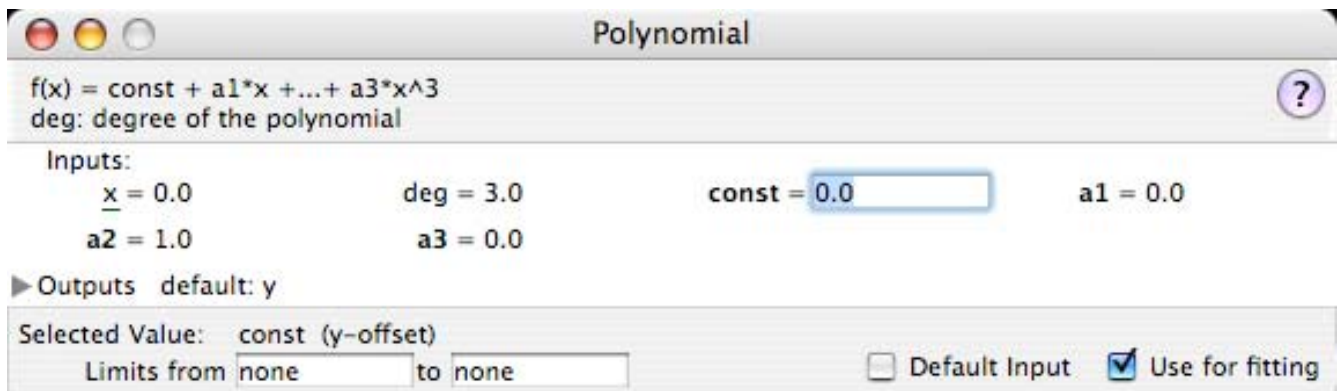
This window determines the behavior of the function. It specifies the *default input* and the *default output*. In this case the default input is the standard one, *x*, but other inputs can be made to be the default one by checking the "Default Input" check box, using the contextual menu that appears when you control-click in the window, or using the hierarchical menu at the top of the Func menu (when the output values are not displayed in the window, there is also an additional shortcut to do this: simply click and hold on the name of the default output and select a different one in the menu that pops up). When the standard x-value is used as the default input, it is possible to use some additional optimizations that can make calculations of some functions faster. A list of all function outputs appears when the disclosure triangle to the left of the word "Outputs" is clicked. For the polynomial function there is only one output, but for other functions, like the Bessel functions, there can be many outputs. Any of the outputs can be made to be the default one using the "Default Output" check box, the contextual menu that appears when you control-click in the window, or the hierarchical menu at the top of the Func menu.

The default input and the default output are used for all operations where a function must be used to transform one value into another, like when a function is plotted in a two-dimensional graph, or when it is used to fit a two-dimensional data set. Some other operations that require more than one input (like

contour plotting) or provide more than one output (like tabulating a function) define their own sets of inputs and outputs and the default input and output set in the parameter window are not used.

Note that whenever the default input is used for any operation outside of the parameter window, its value is varied in that operation, and the value entered in the parameter window is ignored.

Depending on how the function is used, the inputs are often divided into two distinct populations. The ones that are varied according to some pre-defined rule in order to obtain new output values, and the ones that are only changed rarely to influence how the function calculates its outputs or that are determined by some optimization procedure like curve fitting. One can describe in general the first set as the *independent variables* of a function and the second set as the *parameters* of a function. Which group of inputs must be considered as independent variables and which one must be considered as parameters depends on the kind of operation that is performed using the function and can be changed.

But in the simplest, most usual case the default input value is *x*, the default output value is *y*, and there is an additional set of inputs a_1, a_2, .. , a_n that are treated as parameters:

$$y = f(x, a_1, a_2, .. , a_n) \tag{1}$$

For the polynomial function whose parameter window appears above, the input *deg* sets the degree of the polynomial, while *const* and *a1* are parameters that can be determined, e.g., by matching the polynomial function to an existing set of *x* and *y* values (curve fitting). Note that even though any input can be made the default input and it can therefore be transformed into the default independent variable in the above sense, the standard *x* value cannot be used as a parameter (in curve fitting for instance). This limitation ensures that all pro Fit functions have a standard *x*-value that plays the same role consistently, and it allows for a special treatment of the standard *x*-value that allows to optimize the functions for execution speed.

Because the standard *x*-value is indeed most often used as an independent variable, while the rest of the inputs most often play the role of parameters, we will often use the word *parameters* to loosely describe all named input values that usually play the role of parameters as explained above. This reflects the most common usage. However, keep in mind that *any* input can be in principle the default input, and play the role of the independent variable.

An entry in the Func menu makes it possible to save the current inputs under various names to be used later or as the default set for a function. The Parameter Sets submenu in the Func menu allows to save this data for the current session, permanently (in the pro Fit preferences file), or even as an attachment to a function definition file. Whenever pro Fit finds Parameter Sets that can be used for a function, it makes them available in the Parameter Sets submenu.

pro Fit has a set of built-in functions, that you can use "as-is", and gives you the possibility of defining your own functions (there are also many external functions that are distributed with pro Fit and that are ready-to-use). To define a new function, you can use the built in programming language (see Chapter 9, "Defining functions and programs"), or you can write your functions in your own compiler and import them as plug-ins (see Chapter 10, "Working with plug-ins").

Editing in the parameter window

Input values   Click any value and edit it. Use tab or enter to move to the next value, use shift-tab or shift-enter to move back. When an input is selected, the area at the bottom of the

window displays the full name of the input and some other properties: The interval of allowed values (limits), if it is the default input, and its fitting mode.

Output values You cannot edit output values, but you can select them and move from one to the other in the same way as you do for inputs. When an output is selected, the area at the bottom of the window displays its full name and if it is the default output.

copy, paste You can copy and paste input values between the parameters window and data or text windows using the Copy, Cut, and Paste commands from the Edit menu. If you choose Copy with no parameter value selected, all parameters are copied to the clipboard, separated by tabulators. If you choose Paste with no parameter value selected, the text on the clipboard is assumed to contain several values separated by spaces, tabs or carriage returns, which are then used to change all parameter values. You can also paste a list of tab-delimited value while a given input is selected to replace its value and the following ones.

limits All inputs but the standard one (x) can have upper and lower limits, which are displayed in the footer of the Parameters window. These limits are used to constrain the parameter during fitting and function optimization. To change a limit, select the input and enter the limit in the corresponding field. To remove a limit, select the input and clear its limit.

fitting mode To change the fitting mode of a parameter, check or uncheck the option "Use for fitting" in the lower right part of the window. If you check this option, the parameter will be varied during fitting and optimization, otherwise it will be kept fixed. The fitting mode of a parameter can also be recognized by the typeface used for its name. Parameters with names displayed in **bold face** will be varied during a fit. Parameters displayed in normal type face are kept fixed during a fit. As an alternative to using the "Use for fitting" checkbox, simply click the name of the parameter to toggle its fitting mode.

If you define your own function, you must keep the predefined input x as the default input if you want to use the procedure `first`. If you choose another input as the default input, its value would be undefined in `first`. (See Chapter 9, "Defining functions and programs", for more details.). If you plan to use other inputs as the default function input, do not use `first`.

Using functions

The following explains what you can do with functions. It does, however, not describe how to plot or fit functions – these topics are covered in Chapter 8, "Fitting" and Chapter 7, "Drawing" – or how to define a new function – a topic covered in Chapter 9.

Viewing function outputs

To see all the outputs of a function, click the triangle to the left of the word "Outputs" in the parameter window. The list of output values will be updated automatically whenever an input value is modified.

Calculating output values

The first, direct way to observe how a function's output values change in response to a change in its input values is to display all the outputs in the parameter window by clicking the disclosure triangle to the left of the word "Outputs" in the parameters window.

You can also calculate the default output of the currently selected function for a given value of the default input by choosing **Analyze…** from the Calc menu.

Choosing **Tabulate** *Function*(**x**)**...** allows you to create a table of the function's output values in a data window. You are prompted for the first and last value of the table and its step width. The input value that is varied to generate the table is set in the dialog box that appears, and the table will contain all output values.
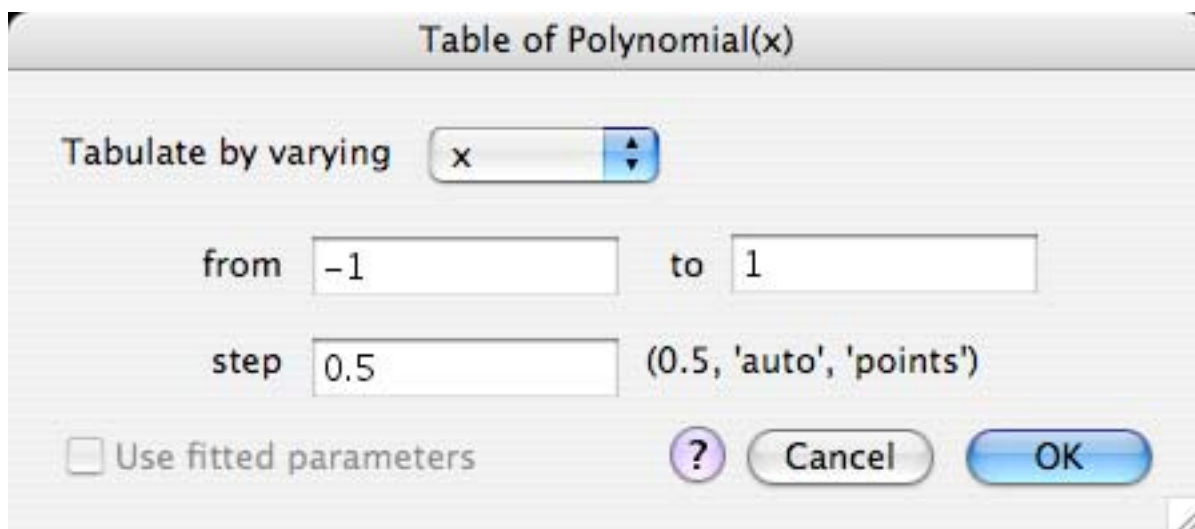
If you enter a numerical value for Step, the function is calculated at equidistant *x*-values. If you enter 'auto' in the field 'Step', pro Fit chooses the *x*-values at which the function is calculated by using a special algorithm that decreases the distance between calculated points wherever the function is strongly bent. In this case only the default output value is calculated.

To tabulate the function at the values of the *x*–column of the current data window, enter 'points' in the field 'Step'. Also in this case only the default output value will be calculated.

Instead of 'auto' or 'points' you can enter a single 'a' or 'p'.

Optimization of functions

The command **Optimize** from the Calc menu lets you find the maximum or minimum of a function by varying any of the input values. To select which inputs must be varied to optimize the function, set their fitting mode to active and use the check boxes in the dialog box that appears when choosing the Optimize command..
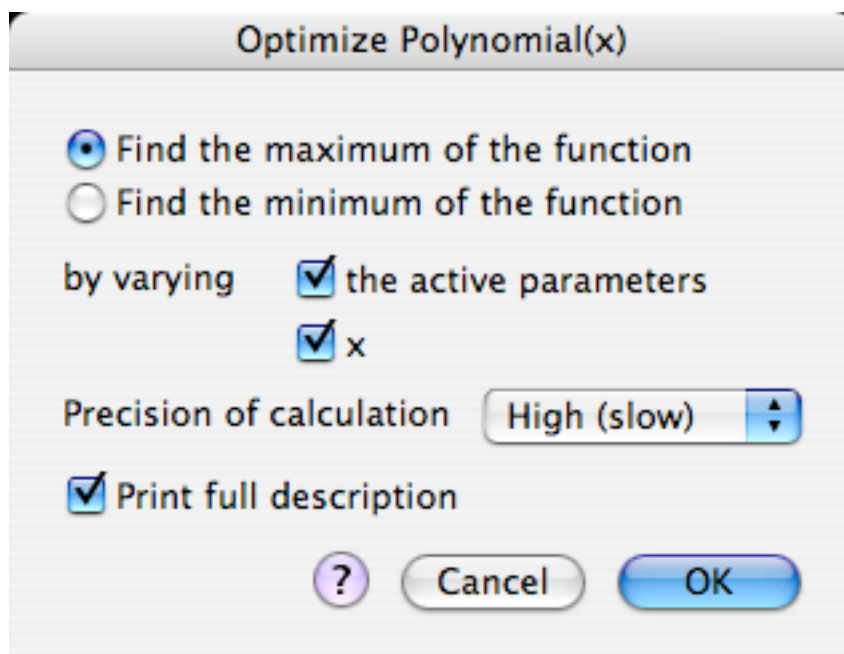
If you check **the active parameters**, the algorithm will vary all parameters that are presently marked as active (i.e. which in the parameters window have a bold face name and "use for fitting" checked). The parameters are only varied within their limits, if such limits are specified.

If you check **X**, the algorithm will vary the function's standard x input. Otherwise the x-value is kept fixed at the given value.

The settings under **precision** affect the accuracy and speed of the calculation. If your function is slow, you should first choose a low precision and, once you are satisfied with the results, choose a higher precision.

Print full description controls the amount of information to appear in the results window.

Note that the command "Optimize" is designed for multi-dimensional optimization. If you only want to vary the function's x-value but not its parameters, you should use the faster command "Extrema" described below.

Analyzing functions

The **Analyze** in the Calc menu brings up a window to calculate the roots, the extrema and the integral of a function:
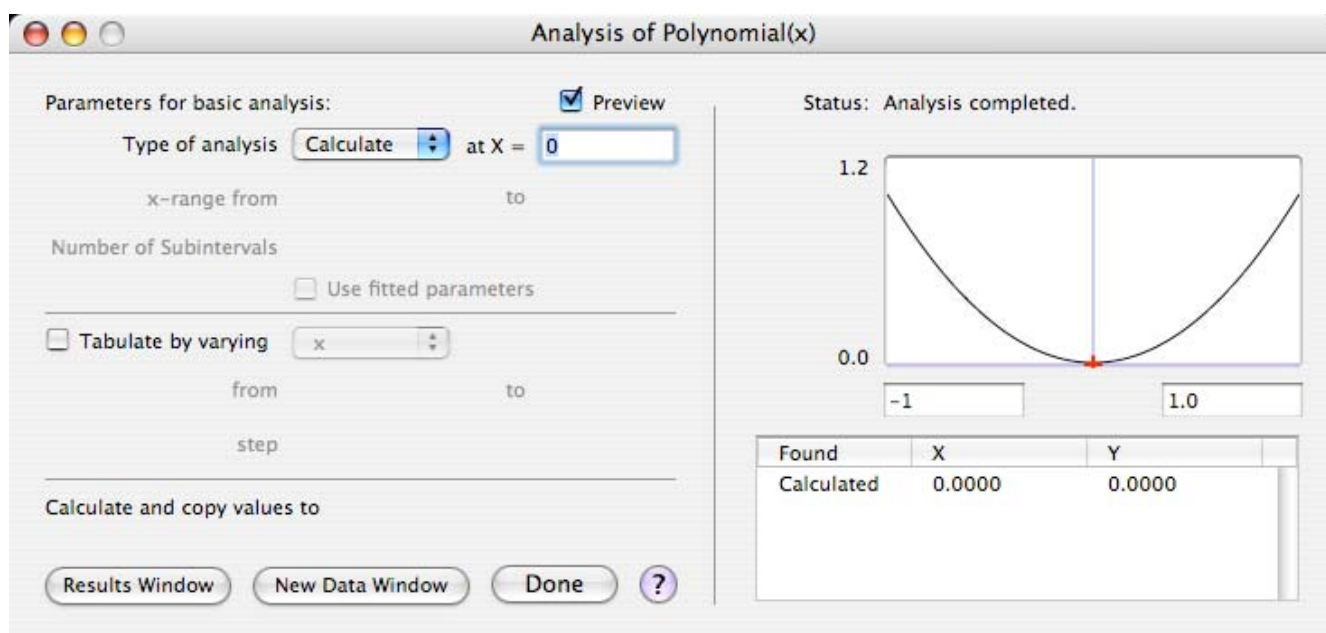
With this tool you can do different types of analysis. A preview area to the right helps you to find the answers you are looking for. While the analysis window stays open, you can modify the functions parameters in the Parameters window, or even change the current function and observe the effects on the results shown in the analysis window.

According to your selection of type of analysis, the result of the analysis is 'immediately' shown in the preview area (i.e. the right half of the window), in the function display as well as in a results list. If your function is too slow, you can switch off the preview check box.

The top part on the left side of the window lets you define the basic parameters for the desired analysis:

- **Type of analysis**: The type of analysis: Calculate a function value, find a root, a maximum or a minimum, calculate an integral or a derivative.

- **X-range from - to**: This defines the range where to look for roots, minima, maxima, or the integral boundaries.

- **Number of Subintervals / Number of Iterations**: To find a root, a minimum or a maximum, the number of subintervals defines how many potential results can be found within the X-range. When calculating an integral, the number of iterations defines (indirectly) the precision of the integration

- **Use fitted parameters**: Check this if you want to use the parameters of the last fit of the current function. Otherwise, the parameters from the parameter window are taken.

The middle part on the left side of the window lets you define a variation of either x or a function parameter for output in the Results window or in a new Data window (i.e. without preview):

If you check **Tabulate by varying**, the analysis is repeated for a number of different x or parameter values. The parameter popup menu lets you select the parameter to be varied.

- **from - to**: This defines the parameter variation range.

- **step**: This defines the size the parameters varyied step by step. When the type of analysis is to calculate function values, you may define 'auto' or 'points' instead of a real number in this entry field.

The lower part on the left side of the window lets you generate and output the desired analysis values:

- Click **Results Window** for doing the complete analysis and writing the results into the Results window.

- Click **New Data Window** for doing the complete analysis and listing the results in a new Data window.

To close the window, hit the button Done, click into the close control of the window, select the Close command from the File menu, or press the 'Esc' key or 'Cmd-.'.

Note: If you want to find the extrema of a function by varying not only its x-value but also its parameters (multi-dimensional optimization), use choose "Optimize" from the menu Calc. This command is described above.

The Spline function

There is one special function in the list of predefined functions: the **Spline** function. The Spline function is defined as a smooth cubic Spline curve going through all your data points. The Spline function is useful for interpolation, especially when you do not have a mathematical model for your data. This is a simple data set together with its Spline function.
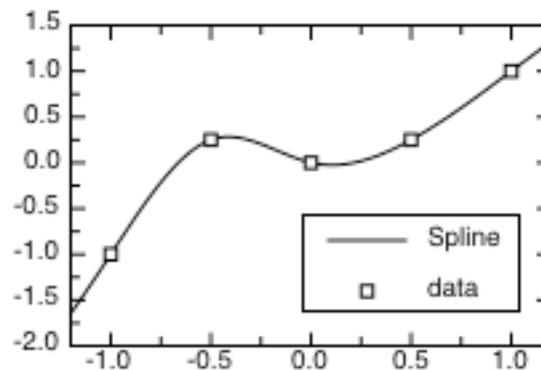
To use the Spline function for a given data set:

1. Choose the Spline function from the Func menu.

2. Bring its parameters window to the front and click the "Spline Settings" button.

A dialog box appears:

Spline Settings

The set of coordinate points used by the Spline function is given by:

○ Current data set

● Data Window   Table 1   ⬍

   X Column   • x–Value   ⬍   Y Column   • y–Value   ⬍

○ Function Parameters

   Number of points   0

   ☑ Set x–coordinates to equidistant values in the current range

                   (Cancel)   (OK)
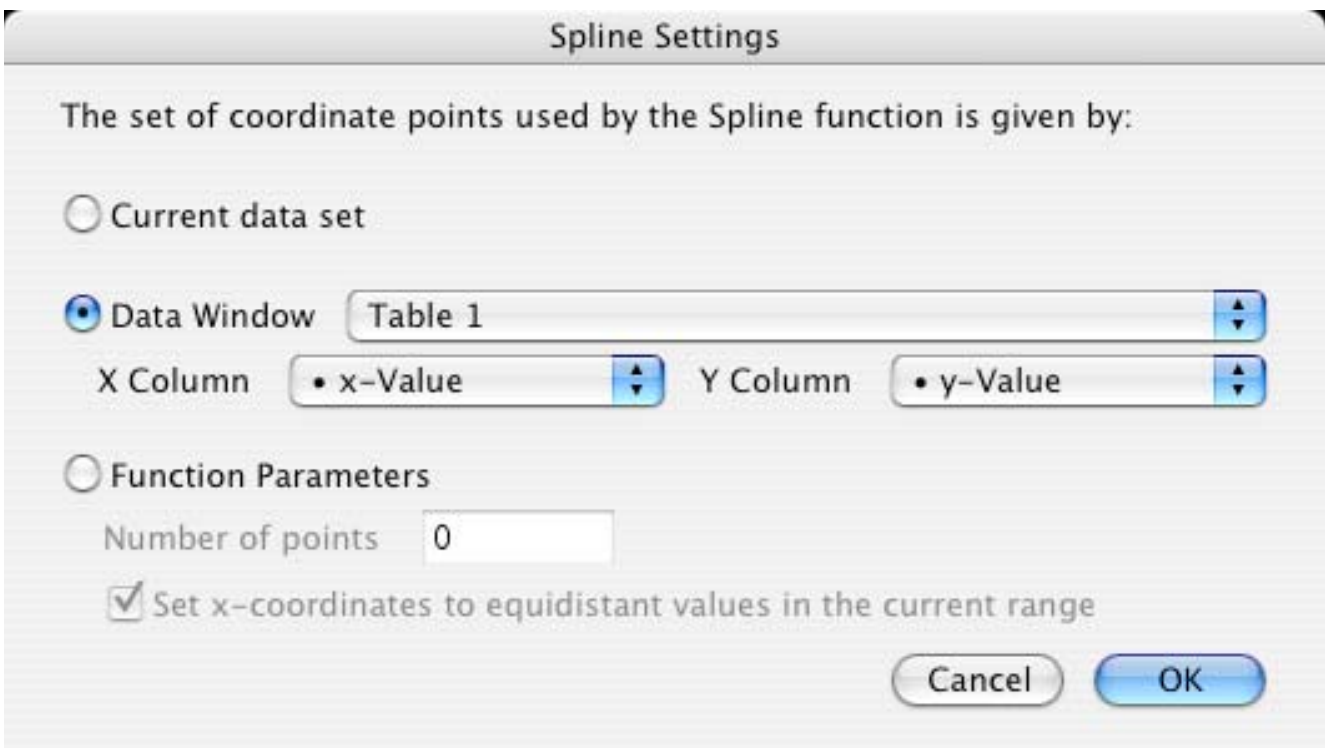
Check **Current data set** to use the x- and y-column of the front most data window. It will then use the set of data points (x_i, y_i) in a data window, where the x_i column and the y_i column are identified by small 'x' or 'y' labels in column head (change the default x- and y- columns by clicking the header of a column while holding down the command key.).

Check **Data Window** to use another data set from a data window.

Check **Function Parameters** to use the parameters of a function as x- and y-values. This allows, *e.g.*, to fit a Spline function to some set of noisy points, in order to get a smooth guide-to-the-eyes curve. When doing this, be careful not to choose too many points for Spline-definition. Other types of functions that are useful to draw a smooth line through a set of noisy points are available as external functions, as part of the pro Fit distribution package.

> If you do not use the "Select Data" button in the parameters window, then the Spline function will use the data in the front most data window (Select the appropriate x- and y-column by clicking the desired column number while holding down the command key).
>
> If you did use the "Select Data" button, but you close the data with the data set used by Spline, then the Spline functions reverts back to using the data set in the front most data window.

6   The Preview Window

There are generally two different approaches that are used by plotting applications for managing graphs and the data used to generate them:

- The first one consists in maintaining a permanent link between the data you plot and the result of the operation (the graph). In this approach whenever you edit the data you used for creating the plot, the plot automatically changes to reflect the new values of the data set. Since the link between data and plot needs to be maintained, it is in generally not possible to save data and graphs separately, and they must be saved in the same document. In applications using this approach, the graph is only a different "view" of the data, but does not lead an independent life.
- In the second approach, graphs and data are independent. Although a graph can be created from data, and data can be recovered from a graph, the two documents lead separate lifes. After it has been created, the graph does not know anymore about the origin of the data used to create it, and if you modify that data, the graph remains untouched.

pro Fit uses the second approach while still providing a link between graphs and data that allows to update a graph easily when the data changes. pro Fit has separate data documents and drawing documents. From the data you can create graphs. From the graphs you can recover the data used to plot them. Drawing and Data documents can be stored and maintained separately and don't automatically affect each other. In Chapter 7, "Drawing", you will see how you can use the Draw menu to plot a function and a data set, obtain graphical representations of your data and functions, and edit the graphs to obtain the precise graph style you are looking for.

There is an ongoing discussion between the supporters of the first approach outlined above and the supporters of the second approach used by pro Fit. A link between data and its graphical representation is in fact also useful, and pro Fit provides it as a convenience, but a real-time, constantly updated representation of data and the current function is provided in a dedicated window, the *Preview Window*.

The preview window is a graphical representation of the current function and/or the current data set. It gives you a graphical "view" of the function and the data set. Any change in the data set or in the function is reflected in the preview window. You can even use the preview window to *graphically edit* the function parameters or the data set.

Use the preview window to have a "quick look" at a function or a data set without actually plotting it. For instance, you can let the Preview Window be a floating window and keep it in front while you load many different data files. The preview window will automatically display all data contained in the current x- and y- columns of the front window.

You can also use the preview window to view functions, graphically edit function parameters, select a range of data points, compare a function to a data set, etc. Functions that have multiple outputs can define *groups* of outputs that are displayed at the same time in the preview window if the default output is part of the group. The other outputs in the same group are displayed in a lighter color. Examples of this behavior are seen when using one of the built-in Peaks functions.

Choose **Preview** from the Windows menu to see pro Fit's Preview Window. This is how the Preview Window looks like when it has its smallest size and is not working as a floating window.
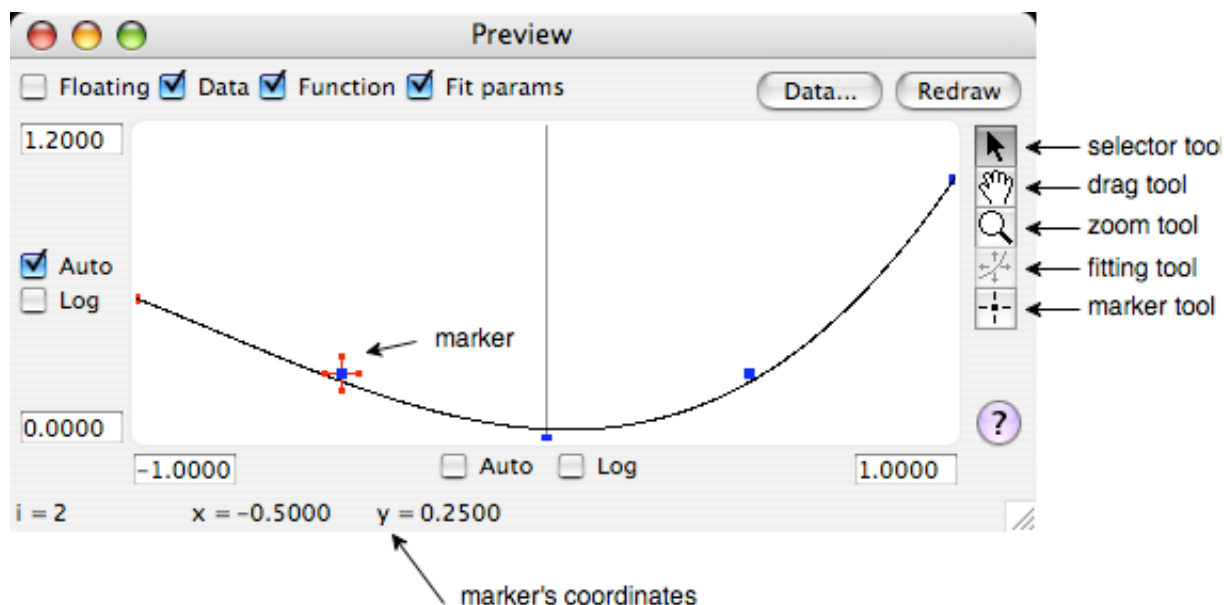
On the left side of the Preview Window there are some check boxes that determine how the window behaves and what it shows. The main part of the window is a rectangular view port that shows a graphical representation of the current function and data set. On the right of the window there is a tool palette with tools for changing the coordinates displayed by the view port, for graphically editing the function and the data set, and for determining precise x- and y- coordinates.

Check **Floating** to make the preview window a "floating window" which always stays in front of all document windows. Uncheck this option to transform it into a normal window, which you can be hidden by other windows.

Check or uncheck **Show data** and **Show function** to choose what is shown in the window. When Show data is checked, the window displays the current data set, *i.e.* the x- and y- columns of the current data window. You can select the data set to be shown in the preview window by clicking the **Data** button or by directly setting x- and y- columns in the data window

The **Fitted params** check box appears whenever a fit was successful, to give you the option of seeing a plot of the function using the parameters obtained in the last fit, instead of seeing the function with the parameters shown in the Parameters window.

Click the **Redraw** button if you want to let pro Fit redraw the complete function at maximum resolution. pro Fit automatically decreases the resolution at which it draws the function if it notices that the function is too slow. You can override this by clicking the Redraw button.

The **Undo** button appears only when the Preview Window is floating, and it allows you to undo the last operation. When the Preview Window is not floating, you can undo the last operation as usual, by choosing Undo from the Edit menu.

At the right end of the title bar there is a **zoom box**. Click it if you want to work with a larger window.

At the edges of the rectangular view port that displays the function and the data set are four edit-fields giving the coordinate range to be displayed. You can edit the values to change the x- or y- range. Between these edit-items there are check boxes labeled **auto** and **log**. Check them to let pro Fit

automatically recalculate the ranges based on the ranges of current function and data set, or to use logarithmic scaling.

There is a **permanent link** between the preview window and the data or function it displays. The preview window always displays an up-to-date representation of the current function and data set. Change a coordinate in the data window, or add a data point, and the corresponding point will automatically appear in the preview window. Change a function parameter and the representation of the function in the preview window will be updated automatically. Modify a function definition and add it to the menu once again, and the preview window will automatically display the new function.

If you select data points in the preview window, the corresponding rows are selected in the data window. If you select some rows in the data window, the corresponding selection is shown in the preview window. There is even the possibility of clicking and dragging data points in the preview window. Doing so changes their coordinates in the data window.

Preview Window Appearance

You can set the color and appearance of data points, function curve, and markers by choosing "Preferences..." from the File menu.

Preview Window Tools

To the right of the preview window there is a palette of five different tools. You can use them to select data points and change their coordinates graphically, to change the ranges of the preview window view port, to graphically change the value of the function parameters, and to set coordinate markers**.**

Selecting data points with the arrow tool

Use the **arrow tool** to select data points. Simply click a data point to select it. Click and drag to select a range of points with a selection rectangle. Hold down the shift key to add points to the current selection, or to remove points from the current selection. If you hold down the **shift** key while dragging a selection rectangle, the selection state of the data points contained in the rectangle toggles between selected and not-selected. Hold down both shift and **option** keys to always add the points inside the selection rectangle to the current selection.

You can set the **color** of the data points and the color used to mark selected data points using the **Preferences...** command in the File menu. If you have a monochrome monitor, pro Fit will use a dithered pattern to mark the selected points.

Whenever you select a data point in the preview window, the corresponding row is selected in the data window. If you then choose Data Transform... from the Calc menu, you can perform calculations on the data in the selected rows only.

Selecting a data point in the preview window always selects the *whole* corresponding row in the data window. If you select a range of data points in the preview and then delete them, you will delete all data in the selected rows and not only in the current x- and y-columns

Changing the ranges of the preview

You can change the ranges of the preview either by editing them manually, or by using the **drag tool** or the **zoom tool**.

Click in the view port area with the drag tool and drag the area of the data set or function curve displayed by the preview. The ranges of the preview will change accordingly. You start dragging inside the view port, but you can go on dragging also outside, thus changing the coordinates by a large amount.

Click in the view port area with the **zoom tool** (the lens) to zoom in and magnify the clicked area. Hold down the **option** key while clicking to zoom out.

If you hold down the **command** key you can click and drag with the zoom tool, thus selecting the precise area that will be displayed in the view port after zooming.

Dragging the function curve

Select the **fitting tool** and click in the view port. Hold down the mouse button while you move the mouse. The curve of the function follows the position of the mouse while the selected function parameter is adjusted accordingly.

When using the fitting tool, you must specify which parameter you want to vary. You can do this either by clicking it in the parameter window, or by choosing its name from the small popup menu that appears below the tools palette in the preview window. You can only vary one parameter at a time.

When you select the fitting tool and click into the preview, the selected parameter is varied until the function curve goes through the point indicated by the mouse. proFit does this by numerically solving the function $f(a,x)=y$, where a is the selected parameter and (x,y) is the point indicated by the fitting tool. If it is mathematically not possible for the function to go through that point, no matter what the value of the selected parameter is, then you will not be able to drag the function curve to that point. The same applies if proFit fails to find numerically the correct value for the parameter.

If you use the fitting tool with a slow function, proFit will automatically reduce the resolution with which the function is drawn, so the function will not appear to be smooth anymore. The resolution will be increased again once you are finished dragging. Click the Redraw button to achieve the maximum resolution.

Inspecting and editing coordinates

The last tool in the tools palette can be used to place **coordinate markers** on a given data point, or on the function curve. Select the marker tool and click the curve or a data point. proFit creates a new marker at the indicated position

While you move the marker tool around inside the view port of the drawing window, the corresponding coordinates are displayed in the bottom left corner of the preview window.

When you create a new marker, it becomes the **active marker**. The active marker is always flashing on and off.

You can create any number of markers. The first marker you create is the **reference marker**. Subsequently created markers are auxiliary markers and are numbered starting from 1. Their number appears when they are active (when they are flashing).

To set the color of the reference marker and of the auxiliary markers, choose Preferences... from the File menu. If the reference marker cannot be distinguished by its color, pro Fit automatically draws it larger.

Marker coordinates are displayed in the bottom left corner of the preview window. If there is more than one marker, there can be two other coordinates displayed to the right of the marker coordinates. They correspond to the distance between the reference marker and one of the other markers.

What the coordinates mean:

	`x,y` are the coordinates of	Δx,Δx are the distances from
No active markers around	the reference marker	the other marker (if there is only one)
One active marker	the active marker	the reference marker

If a marker is active, its coordinates are displayed in **editable** fields. Edit any of these fields to set the coordinate of the marker.

If the marker is a data marker and the preview window is big, the data window row number that corresponds to the marked data point is also displayed. It is found above the x-coordinates and is labeled "`i = `"

> The behavior when changing the text in the edit fields containing the marked coordinates varies depending if the marker is on a data point or if it is on a function curve.

- If the marker is on a data point, the coordinates displayed in the edit field correspond to the coordinates of that data point in the data window. Editing them changes the values in the data window.
- If the marker is on a function curve, editing the coordinates sets the position of the marker. If you edit the y-coordinates, pro Fit numerically inverts the function to find the corresponding x-value. You can use this feature also as a shortcut to calculate the inverse of a function, or its root.

Coordinate markers can be accessed from pro Fit programs using the predefined functions `GetMarkedX`, `GetMarkedY`, and `GetMarkedCoords`.

Managing coordinate markers

We already saw above how to create markers and look at their coordinates. There are a few other simple operations that can be applied to markers.

- Click a marker to make it active.
- Click a marker while holding down the option key to transform it into the reference marker
- Hit the delete key (backspace) while a marker is active to delete it.
- Click and drag a marker to move it to a new position.
- Move a function marker to the right or left border of the viewport to delete it.

To move a marker, click and drag it, or use the left and right arrow keys. A data marker jumps to the next point to its left or its right, a function marker will move along the function curve. pro Fit makes sure that you don't move a marker outside the ranges of the viewport. You can override this by holding down the option key while moving the marker with the arrow keys.

When you have markers on the function and you uncheck the show function checkbox, all of them are deleted. The same applies to markers on data points when you uncheck the show data checkbox.

Uncheck and check the show function and/or show data checkboxes if you have many markers around and want to get rid of all of them in one rapid move.

Data markers store their position as the number of the data point they mark. If you have data markers around and you delete or add points to the data set, the data markers might move to a new data point. If no new point corresponding to the old index is found for a given marker, that data marker is destroyed.

If you have function markers and you change the ranges of the display in such a way that their x-coordinates are not visible anymore, those markers are destroyed.

Tips and tricks

Using the preview window during a fit

If Show function is checked during a fit, the function is redrawn from time to time to show how it changes during the fit. This lets you monitor how well the fit converges. However, drawing the function takes time. You should close the preview window or uncheck Show Function to obtain the fastest fitting. The same applies to the preview that is built into the fitting dialog box.

The same thing happens when you use the Error Analysis feature. To perform error analysis, pro Fit generates random sets of synthetic data points and fits the function to it. If Show Function is checked in the preview window, you will see how the function curve varies in correspondence to the fitted parameters.

See Chapter 8, "Fitting", for more details on the fitting process and the Error Analysis algorithm.

Choosing initial values of function parameters

You can display the data you want to fit together with the fit-function in the preview window. You can then use the fitting-tool to drag the function in such a way that it follows the data points as closely as possible. Try using the fitting-tool with the various parameters you want to fit.

This is a kind of "hand fitting" that can be a very useful and fast way to set up a reasonable set of starting parameters for a fit.

For special applications, you can also mark certain features of your data set using coordinate markers and write a small program which reads the coordinates of these markers and uses them to calculate the optimal initial values for the parameters of the current function.

7 Drawing and Plotting

Drawing and plotting takes place in a *drawing window*. This window supports most features of commonly used drawing applications.

We will first describe the drawing window and its general features.

The section "Drawing" discusses standard drawing objects and editing techniques.

The section "Plotting" is devoted to the plotting commands used to produce graphical representations of your data and functions. It discusses how to manage graphs and how to edit them.

The drawing window

A drawing window always contains one single page. You can select its size and orientation by choosing Page Setup... from the File menu. Before choosing Page Setup, make sure that you have set up your printer.

A dotted rectangle frames the *printable area* of the page. Objects that lie outside this rectangle do not print. See your printer's manual for more information on printers and paper sizes.

You can view the page in a drawing window using various zoom factors, which you can set using a popup menu in the drawing window tools palette.

Drawing tools

pro Fit provides various tools for editing drawings. These tools are collected in a "tool box", which is either placed in the left margin of a drawing window or in a separate floating window.

To place the tools in a separate drawing window, choose "Drawing Tools" from the Windows menu. The floating tools palette appears. To move the tools back to the drawing window, simply close the floating window.

If you will never want to have drawing tools inside the drawing windows, you can disable this option: Choose "Preferences" from the Files menu and check "Always use floating toolbox" in the "Drawing" panel.

The upper part of the tools palette contains tools that are used to select, move or create simple objects, such as rectangles and text. Then there is a tool that can be used to pick up a color and apply it to another graphic object and a tool that lets you draw the individual data points such as those used in graphs. A further tool is provided for generating control shapes, such as buttons. The rest of the tools palette contains popup menus for setting line styles and fill patterns, and for choosing the zooming factor of the current view in the drawing window. The drawing window can be viewed at zoom factors from 25% to 400%. To learn more about these tools, refer to the section "Drawing" later in this chapter.
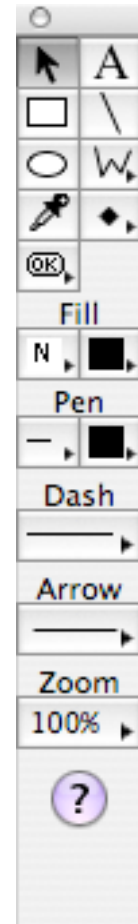
Coordinates, accuracy and drawing info

pro Fit uses floating point numbers to store the size and position of the various drawing objects. This provides a positioning precision that is much more accurate than any output device (printer or monitor). This is important because all drawing objects can also be created by a user-program. If you write a program that produces graphical output, then you are likely to need a high precision coordinate system. pro Fit gives you just this. Any drawing that you generate from a program is produced at very high resolution and it will give optimal results when printed on any output device or when exported to other applications as a picture or a PDF shape. The precise coordinates of any drawing objects can also be viewed after it has been created using the pro Fit *Coords* window, which will be described later in this chapter.

Although all coordinates are precise floating point numbers, apparent accuracy will obviously suffer when drawing on a low resolution device, such as a normal monitor. In order to represent your drawing at a certain resolution, determined by the zoom popup menu, pro Fit must round the floating point coordinates describing a drawing object. In addition, fractional line widths and coordinates can be represented by anti-aliasing effects. As an example, a line that is half-a-pixel wide will be drawn with a light gray. These effects are produced by the systems's *Quartz*drawing engine, and they can be optionally switched off.

When you draw something at a low resolution, pro Fit must figure out reasonable floating point coordinates. It does this by "extrapolating" from the low resolution appearance in such a way that a

high resolution view would give the same symmetry. For example, at the 100% view you can draw three overlapping lines with thicknesses of 0.25, 0.5 and 1.0 pts. All three lines have exactly the same appearance (e.g. they appear 1 pt thick). pro Fit sets up the floating point coordinates of the lines in such a way that the thinner lines are **centered** on the 1 pt thick line.

Thanks to this interpretation you get the same result, at 100% view, if you draw a 1 pt line and then make it 0.25 pt thick, or if you draw a 0.25 thick line directly. On the other hand, if you draw a 0.25 pt thick line at 400% view, go to 100% view, and draw another 0.25 pt thick line on top of it, the two lines will not overlap. This is because the first line was positioned with a much larger precision than the first line. Use the **Align** submenu in the Draw menu to make sure that such lines really overlap, or look at their coordinates using the Drawing Info window (see later).

Likewise, if you have two graphs or rectangles, set their size to be exactly equal, and overlap them at 100% view, one of their borders might be off by one pixel if their *position* is not exactly the same. This is because roundoff errors must occur when calculating their rounded coordinates at 100% view. If you set their position to be exactly equal (using the Align command or using the Drawing Info window), the roundoff errors are exactly the same for the two objects, and they do overlap exactly in the 100% view, too.

If you are concerned with precise positioning, e.g. when drawing overlapping lines or placing arrows on the axes of a graph, always go to a larger zoom factor (*e.g.* 400%) or have a look at the underlying floating point coordinates. You can do this using pro Fit's **Drawing Info window**.

Choose "Coords" from the Windows menu to see this floating window.

Whenever a single drawing shape is selected, the Drawing Info window shows its floating point coordinates, i.e. its size and its position in coordinates that make sense for the particular shape which is currently selected.

The first row of the Drawing Info window gives the absolute coordinates on the paper, the second row gives the dimensions of the selected shape, and the third row gives the angle of its diagonal and its length. The last row shows the current coordinates of the mouse. The units used to display the coordinates can be chosen using the "Preferences..." command.

All the coordinate fields are **editable**. Simply click a coordinate and enter a different number to change the size or the position of the selected shape. For example, you can set the precise length and orientation of a line by entering the corresponding coordinates in the edit fields in the third row.

The Lens check box lets you open a small viewport with an enlarged version of the region around the mouse.
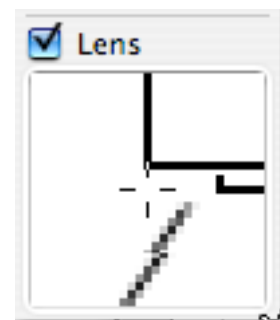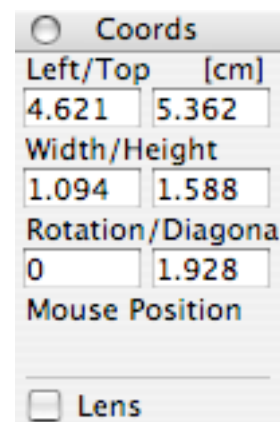
Drawing objects

A drawing contains different *objects*. There are three different classes of objects in a drawing window:
- Objects that are created by choosing **Plot Function** or **Plot Data** from the Draw menu, such as a graph and its associated legend.
- Objects that are created using the tools in the upper part of the **tools palette**, such as texts, lines and rectangles.
- Objects that were created in another application and that are imported as pictures to pro Fit by choosing **Paste** in the **Edit** menu, by importing a picture file or by dragging and dropping them in a drawing window.

The first class of this list (graphs and legends) is discussed in the section "Plotting". The other classes (objects created by using the tools palette and imported pictures) are discussed in the following section.

Drawing

This section describes the general drawing commands and the use of the tools palette.

General drawing commands

General drawing commands apply to all types of drawing objects. These commands are probably already known to you if you ever used any drawing application.

Here we shortly review them one by one.

To **select** an object in the drawing window:

1. **Choose the arrow tool in the tools palette by clicking the box containing the arrow symbol.**

2. **Click the object you want to select.**

   A selected object has four small black rectangles (*selection handles*) at the corners of its enclosing rectangle.

To select **multiple** objects, you can either click on the desired objects while holding down the shift key, or you click into an empty part and drag the mouse to generate a dotted selection rectangle: every object enclosed by the rectangle will be selected. Click on an object while holding down the shift key to deselect it.

To **move** an object:

1. **Click the object and hold down the mouse button.**

2. **Drag.**

If you hold down the **shift** key while dragging, movement is constrained to horizontal or vertical directions. If you hold down the **command** key while dragging, movement is constrained to diagonal (45°) directions.

In MacOS 7.5 and later, or if you have the Drag and Drop extension installed, you have a few more options available:

- If you hold down the **option** key while dragging, the object is **duplicated**, i.e. a copy of the original is created at the destination instead of simply moving the original.
- You can drag one object from one drawing window to another. If you do this, a **copy** of the object is created in the destination window.
- You can drag objects to any other application (supporting drag and drop), or to the Finder's desktop. In the latter case the Finder will produce a small picture clipping, which you will be able to use later on, either by dragging it back to a pro Fit window, or by using it in another application.
- You can drag objects into the Trash to delete them.

To **change the size** (**resize**) of an object:

1. Select the object.

2. Click into one of the four black selection handles at its corners and drag.

While dragging, the new outline of the object is shown.

If you hold down the **shift** key when resizing, the **proportions** of the object are maintained, or the height or width remains constant. If you hold down the **option** key when resizing, the horizontal and vertical dimensions of the object become equal. If the object is a group of different objects, hold down the **command** key to tell pro Fit to resize all of the objects of the group, regardless of their type (normally pro Fit would not automatically resize texts or data points).

To **rotate** an object:

1. Select the object.

2. Choose the desired rotation from the Rotate submenu in the Draw menu.

All objects except graphs and legends can be rotated by angles multiple of 90˚. Lines, Polygons, and Rectangles can be rotated by any angle, not just multiples of 90 degrees.

To **flip** an object, i.e. to exchange its left and right sides or its top and bottom:

1. Select the objects to be flipped.

2. Choose the desired operation from the Flip submenu in the Draw menu.

"Flip Horizontal" exchanges the left and right side of the objects. "Flip Vertical" turns it upside down.

Note that you can only flip lines and polygons. It is not possible to flip graphs, legends, imported pictures, or text. (Flip has no effect on rectangles and ovals).

To **change the order** in which several objects overlap:

1. Select the appropriate objects.

2. Choose the desired operation from the Send submenu in the Draw menu.

You can move objects one position forward or backward (commands "Forward" and "Backward") or you can bring them to the front or to the back of all other objects in the window ("To Front", "To Back").

To **align** objects:

1. Select the objects to be aligned.

2. Choose the desired operation from the Align submenu.

Using this menu, you can align objects to each other, or distribute them regularly. If the objects are a group of text objects, then every object retains its alignment when you edit it.

To **group** objects:**1.   Select all objects to be grouped.**

2. Choose Group from the Draw menu.

Objects that can be double-clicked to change them (*e.g.* text objects, a graph, or its legend), can also be double-clicked and changed while they are part of a group. You don't have to ungroup them. If the objects are text objects and you aligned them with the Align command before grouping them, their alignment will be maintained when they are edited.Choose Ungroup from the Draw menu to ungroup a group.

> If you resize a group containing text objects or data point symbols, the proportions and size of the text and data points remain the same. If you want to resize them proportionally with the group, hold down the **command** key while resizing the group

Objects created with the tools palette

The upper part of the tools palette contains the drawing tools needed to create some of the more simple drawing objects.

The lower part contains pop-up menus to select background patterns, line widths and dashing, and arrows. Their use is explained in the section "Editing drawing objects".

Text objects

Use the text tool to create text objects:

1. Select the text tool from the tools palette.

2. Click inside the drawing window.

The text dialog box appears:

Edit Text

Electric gradient $\mathcal{E}_{ij}$

Font Helvetica ▶   Selected keyboard 🇨🇭🇩🇪

☑ Use Greek keyboard for Symbol font   ☐ Bold ⌘B
                                            ☑ Italic ⌘I   ☐ Extended
Position -3 ▶ ↕   ☐ Underline   ☐ Condensed

Size 24 ▶ ↕   (?) (Cancel) (OK)
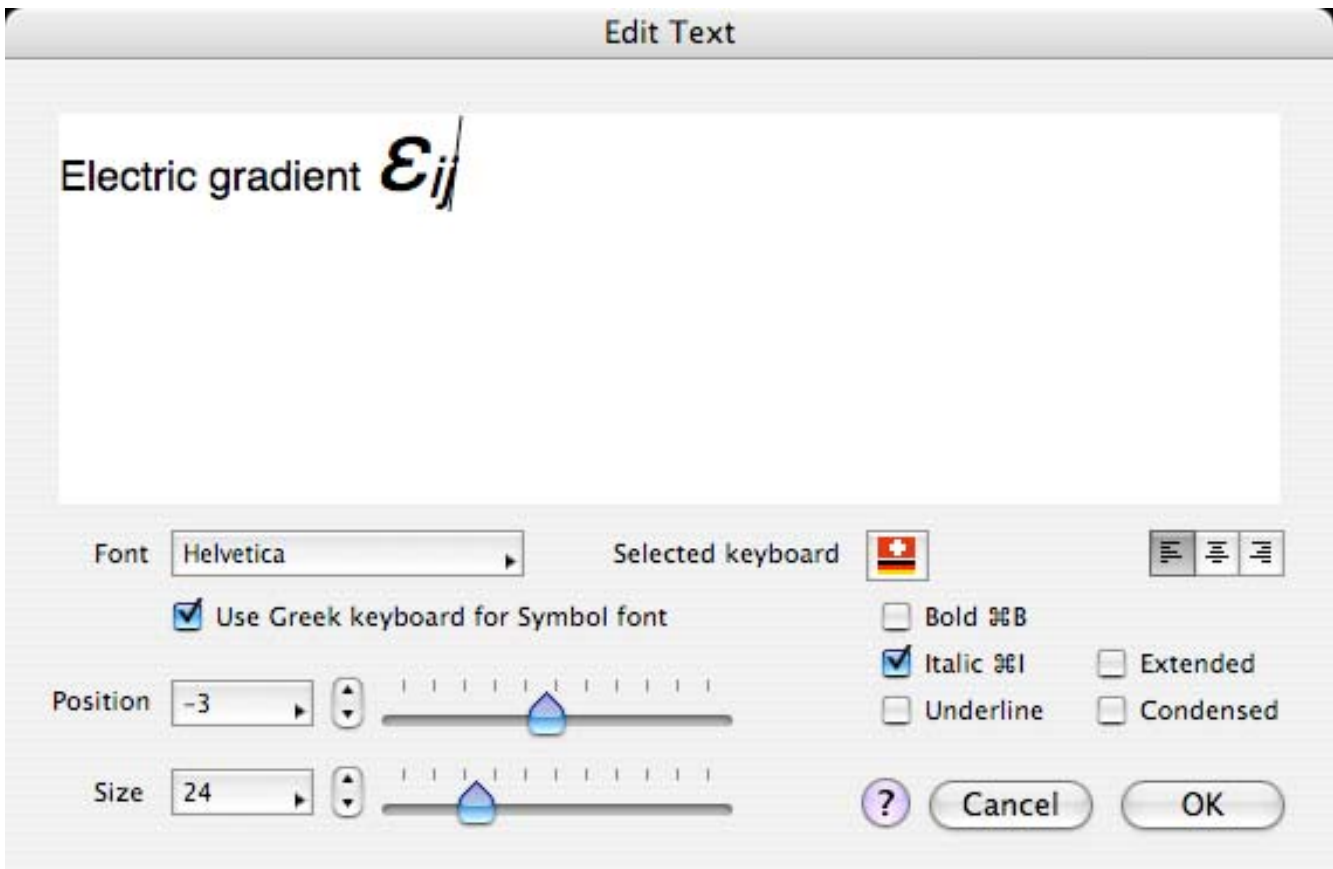
Here you can enter your text and specify font, font size, text styles and the vertical position of each character. pro Fit uses the shift-command key equivalents "H", "T", and "S" for the fonts Helvetica, Times, and Symbol, respectively.

To offset a character vertically, use the controls to the right of the title **Positions**. To change the size of a font, use the controls to the right of the title **Size**. To generate subscript and superscript characters, hold down the command key and hit the up or down arrows – to go back, hold down the command and shift key and hit the space bar.

You can also set the **justification** of a text object (right justified, left justified or centered) by using the align commands in the Draw menu (left, right, center horizontally).

Instead of clicking the **OK** button, hit the Enter key or hold down the command key and hit the Return key. (On a desktop keyboard, the Enter key is the large key at the right buttom of the keyboard. On a laptop keyboard, hold down the fn key and hit the Return key.)

To enter **symbol characters**, such as α β, or γ, switch the keyborad to a layout that allows to enter such characters. For this purpose, it is easiest to enable the Input Menu in the menu bar. This menu is usually represented by an icon carrying a country flag or country specific character towards the right end of the menu bar. If no such icon appears in you menu bar, choose "System Preferences..." from the Apple menu and go to the International preferences pane. There, select the "Input Menu" tab and activate, besides your customary keyboard layout, e.g. the Greek keyboard or the Character Palette. After having done so, you can switch between keyboard layouts by hitting the spacebar while holding down the command key.

pro Fit uses unicode for character representation. If you enter a unicode character that cannot be handled by a given font, pro Fit uses fallback algorithms to find a font that does handle this character. This will usually give the expected results when displaying and printing through the Quartz rendering machine. If, however, you are relying on Quickdraw or Postscript output, we recommend that you always explicitly switch to a font that carries the desired characters. In particular when using symbol characters, such as α, β, or γ, we recommend to switch to the Symbol font.

> If your text object is part of a group object, it is not resized when the group is resized. If you want to resize the text objects within a group, you must hold down the command key while resizing the group.

Rectangles and ellipses

Rectangles and ellipses are created using the corresponding tools of the palette. Select the appropriate tool, click the desired position of one corner of the rectangle (or the enclosing rectangle for an ellipse) and then drag the mouse to the opposite corner.

Note that you can draw partially closed ellipses or circles. To do so, create a regular ellipse or circle and double-click it or choose Shape Settings… from the Draw menu. In the dialog box that appears, you can select the starting angle and arc length of the section to be drawn.

Lines and polygons

Crating lines and polygons is easy as well. Select the appropriate tool, click the start of the line and drag to its end. For polygons, click at the positions corresponding to the corner points of the polygon (release the mouse when moving from one point to the next). Double click when finished.

By holding down the mouse button for a while when you select the polygon tool, you can change the tool to the one for closed polygons

Hold down the shift key to constrain lines (or polygon sections) to horizontal, vertical, or diagonal directions.

When drawing a polygon, hold down the command key and double-click to create a corner that remains a corner even when the polygon is smoothed.

Lines and polygons can have **arrows**. To define at which ends of the line (polygon) arrows must be drawn and to select the type and size of the arrow(s), use the arrow pop-up menu in the tools palette.

To **smooth** polygons:

1. **Select the polygon you want to smooth.**

2. **Choose the appropriate smoothing method in the Smooth submenu in the Draw menu.**

The two possibilities for smoothing can be seen in the figure below. You can either select a standard Bézier curve that does not touch the corners of the polygon, or you can select a smoothed curve that goes through all the corners of the polygon.
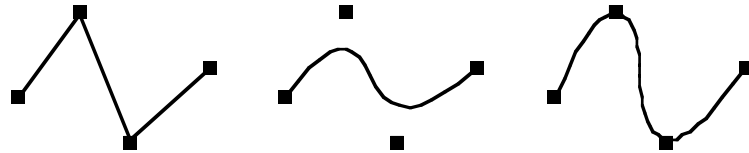
An unsmoothed polygon and its two smoothed versions.

To **reshape** polygons:

1. **Select the polygon you want to reshape.**

2. **Make sure it is in reshape mode.**

   If the selection marks of a polygon appear at the corners of its enclosing rectangle, the polygon is not in reshape mode. If the selection marks appear at its corners it is in reshape mode:
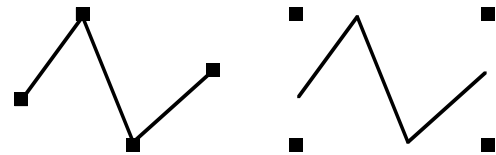
3. **If the polygon is not in reshape mode, choose Reshape from the Draw menu, double-click it, or type the Enter key**

   This puts the polygon into reshape mode.

To **move** one of the corner points of a polygon, click and drag it. To **remove** one of the corner points, click it while holding down the option key. To **add** a corner point, click a line of the polygon while holding down the option key. Note that you can only add points to unsmoothed polygons.

Points

When plotting data, the data points are represented by special symbols. You can create such plot symbols manually anywhere in a drawing window. This is useful for creating your own legends or for exporting single point symbols to other applications (*e.g.* for figure captions).

Since the point symbols can assume a quite large size, they can also be used as parts of standard drawings. Data point symbols can be edited using a particular set of tools that let you achieve effects not easily achieved with other drawing objects (below you will find more details about editing data point symbols).

To create a point object:

1. Choose the point tool from the tools palette.

Keep the mouse button down for a little while to select the symbol that you want to use. A pop-up menu with a choice of data points appears. Its top part contains a set of standard, predefined point symbols. The last line contains user-defined symbols, and the Other... field lets you define new point symbols.
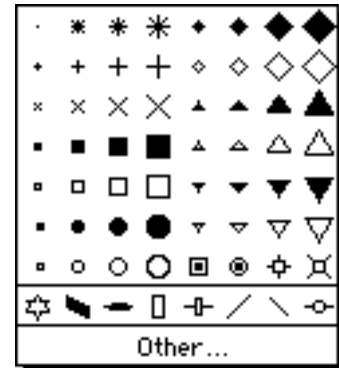
2. Click the desired position within the drawing.

A new point symbol drawing object is created.

1. Choose the point tool from the tools palette.

To change the plot symbol of a point object, select it and choose the desired symbol from the point style pop-up menu in the tools palette (or double-click it to go directly to the custom points dialog box).

If the selected object is a graph or a legend, the new point style is applied to the data plots contained in the graph. See the section 'Graphs and legends', later in this chapter, for details.

On the right you see a selection of the data point styles offered by pro Fit. Choose Other... from the point style menu to create your own data point symbols using the "Custom Points" dialog box:
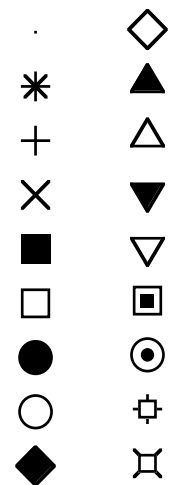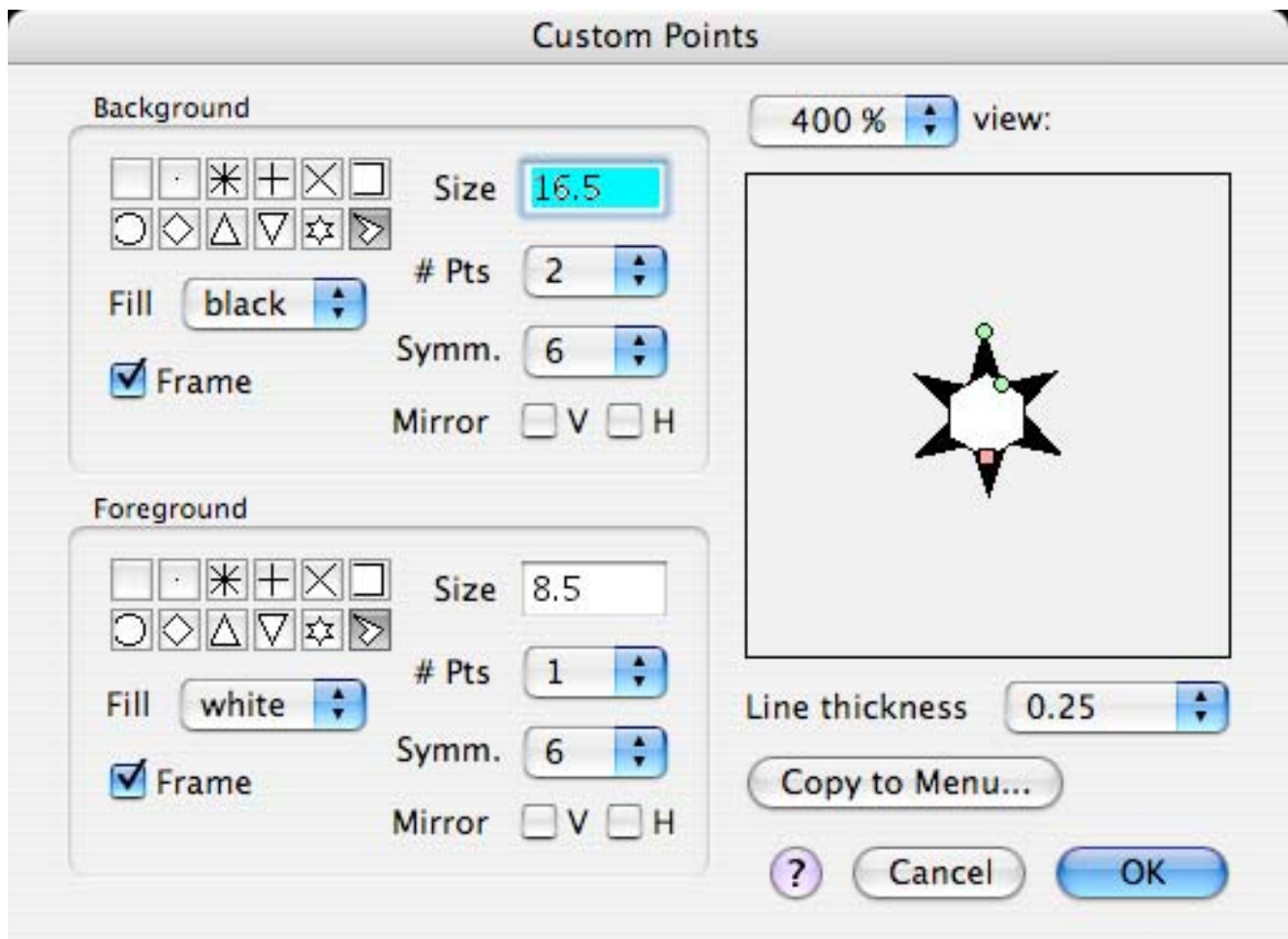
proFit defines a data point symbol as a background shape and a foreground shape. With this dialog box, you can design both of them. proFit offers some predefined simple shapes, and lets you edit any closed polygon to define a new data point symbol. In the above example, both foreground and background shapes are defined using a closed polygon. You can use this dialog box simply to change the size of an existing point symbol, or to design more complicated point symbols.

Draw the foreground and background shapes in the preview area at the right of the dialog box. Use the popup menu above it to set the magnification of the preview. The center of the preview area defines the "hot spot" of the data point symbol. When plotting, the "hot spots" of data point symbols are positioned on the correct mathematical coordinate.

Draw a closed polygon by dragging the polygon handles (the little circles or squares at the edges of the polygon). To make your work easier, proFit lets you define a rotational symmetry and mirror symmetries. Choosing 5 from the **Symmetry** popup menu (like in the above example) tells proFit to draw the definition points at 5 positions 360/5 degree apart before connecting them with lines. Use the **# Pts** popup menu to set the number of definition points. Checking the **H** or **V** check boxes tells proFit to draw the definition points at the 2 positions obtained by mirroring them at a horizontal or vertical axis, respectively. You can achieve quite astonishing effects by combining these symmetry settings and using only one or two definition points.

Hold down the shift key while dragging a polygon handle to constrain the dragging along radial directions. Hold down the command key while dragging a polygon handle to resize and rotate the whole polygon in one single move.

Choose a Symmetry of "1" and no mirror symmetries to draw a polygon free-hand.

> Note that if you do this you can draw a polygon that is not centered inside the preview area. This means that if you use such symbols for plotting, the symbols will not be centered on the mathematical coordinates of the data points.

Click the **Copy to Menu...** button to add the point symbol you just defined to the point symbols menu for later use.

The data point symbols you define are normally used when plotting (see the section "Plotting", below, for more details on this). However, you might also want to use them to achieve some special effect in a drawing. For example, you can use a triangle or a rectangle to define a point, and you can rotate them by any amount. You can't do this that easily using the standard drawing tools. You can also define closed polygons with any special symmetry. The data point symbols you define can then be used as drawing objects in the drawing window (their size can be quite big). You will be able to resize them as usual by dragging a selection handle, and you can always modify them by double clicking them.

Control shapes
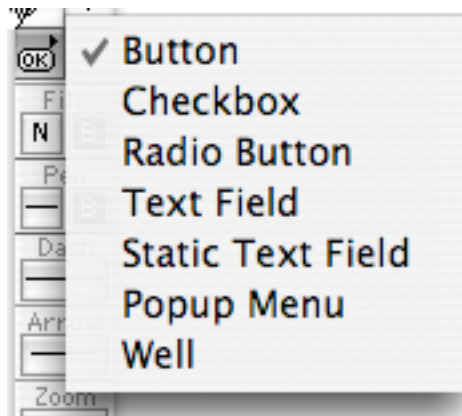
Control shapes allow you to add buttons, check boxes, radio buttons, text fields, popup menus and image wells to a drawing window. These shapes can be accessed by a program for generating complicated dialog boxes.

Chapter 9 of the manual, section "Attaching programs" tells you more about how to use control shapes.

The following are the control shapes that you can used:

Buttons: These are simple objects that hilite when clicked.

Checkboxes: They change their state when they are clicked.

Radio buttons: They are checked when they are clicked. They usually come in groups. The program that manages the radio buttons is responsible for un-checking all other radio buttons when one radio button is clicked.

Text fields: These are objects that contain text. Generally, text fields can be edited. If you don't want the text field to be editable, use a "Static text field".
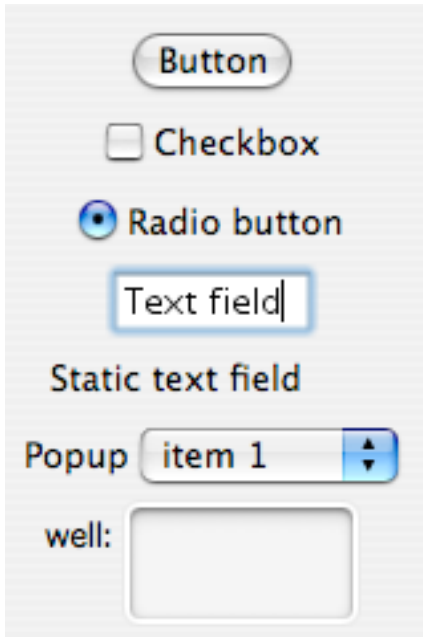
Popup menus: These are objects that have several "values" which can be selected by choosing them from a popup menu.

Wells: These shapes are usually used as background for other objects, e.g. a graph. They consist of a white rectangle.

For a more detailed description on how to use control shapes, see Chapter 9 of the manual, section "Attaching programs".

Editing drawing objects

You can change many attributes of drawing objects, such as color, line thickness or background pattern. To do this, first select the desired object(s). Then change the attributes using the **Fill**, **Pen**, **Dash**, and **Arrow** popup menus.

A fill pattern and a fill color can be specified for all drawing objects, except simple lines. See Chapter 10, "Printing" for a list of limitations on patterns when printing with PostScript.
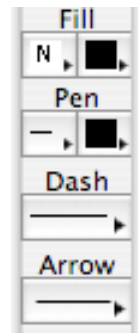
A line color can be specified for all drawing objects except imported pictures.

The two **Pen** popup menus are used to select a thickness and a line color. The dash pattern of a line is selected using the **Dash** popup menu. Choose Other... from this menu to design your own dash pattern and add it to the Dash menu.

The line thickness and dash pattern can be specified for all objects containing lines. If the selected object is a graph, the line styles of the axes, ticks, grid, and frame will be changed. The color also applies to the labels. (More complex options are available for the graph. See section 'Graphs and legends' in this chapter.)

If the selected object is a legend, you can change the appearance of the curves and data sets displayed in the legend and the corresponding graph. See the section 'Graphs and legends', later in this chapter, for details.
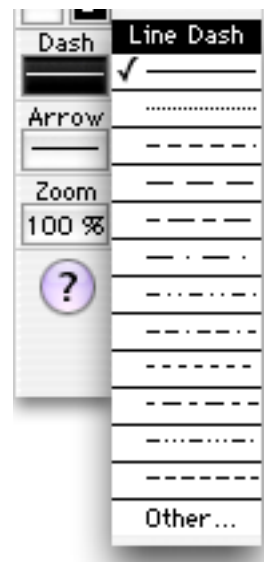
Arrows of various size and shape can be added to polygons and lines using the Arrows pop-up menu.

Arrows can be added to all lines and polygons, smoothed as well as unsmoothed.

Choose Other... from the Arrows menu to design your own arrows and add your personal arrow styles to the Arrow menu.
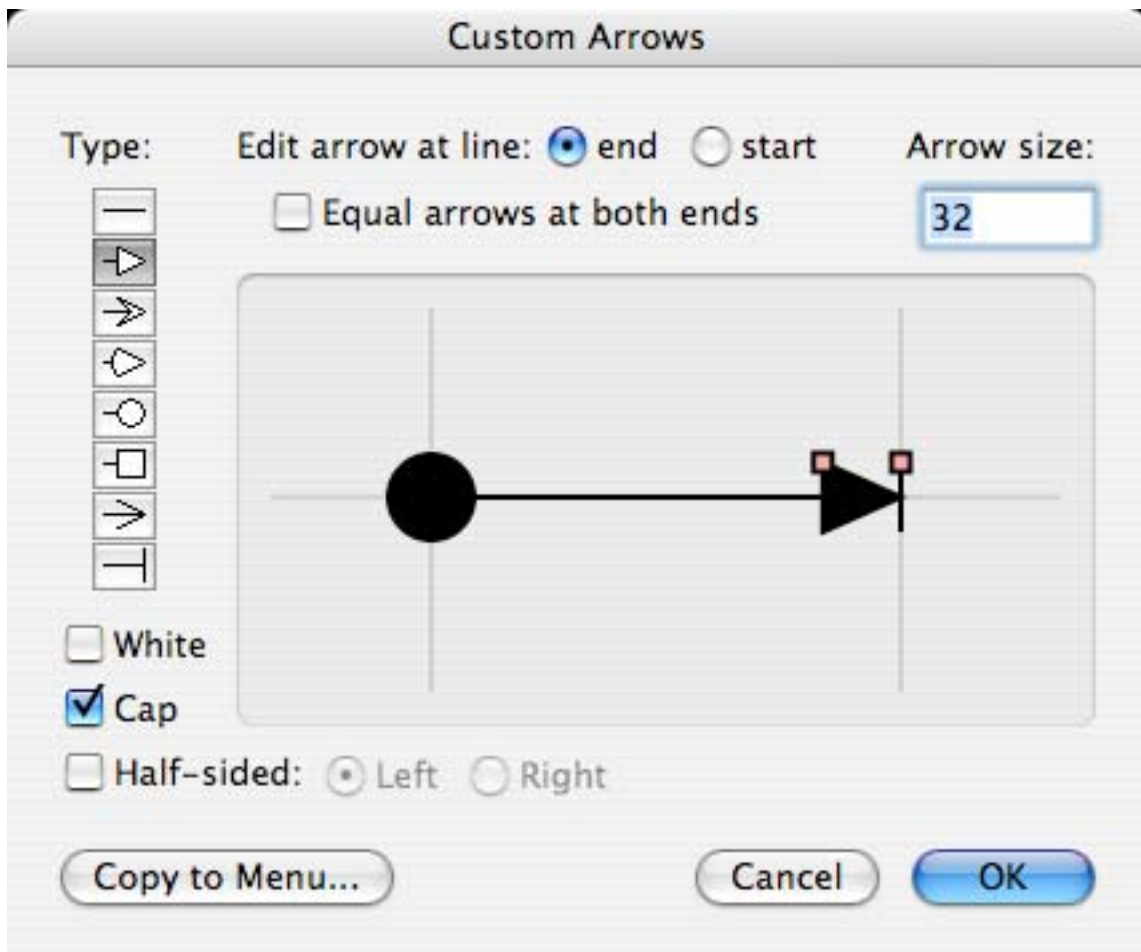
You can define a different arrow to be used for the start and the end of a line, use half-sided arrows, define various other types of line caps, etc.

Fill colors and line colors are set using the corresponding popup menus.

To copy the **line color** from one object to another

1. **Click the color measuring tool (** **)in the tool palette.**

2   **Click the color you want to copy to pick it up.**

   The shape of the cursor changes and becomes a paint bucket.

3   **Click the drawing object to which you want to transfer the color.**

   The line color of the clicked shape takes the color you picked with the color measuring tool.

To picked up a **fill color** for the target shape, instead of a line color, hold down the shift key while clicking with the color measuring tool.

proFit stores the color that was measured with the color measuring tool inside the standard color popup menu. This opens up an other, even more flexible possibility to copy colors.
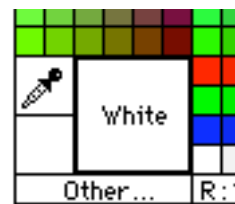
1. **Click the color you want to copy with the color measuring tool.**

2. **Select an object and apply the measured color using the standard Fill- or Line-color popup menus.**

On black and white monitors pro Fit displays a simpler version of the color menus, with a more limited choice of color. To see the standard color menu which is displayed on color monitors, hold down the option key while clicking the popup menu symbol.

Exporting pictures

There are a number of ways to export pro Fit drawings:

- using the Copy or Cut commands in the edit menu,
- dragging them and dropping them to their destination,
- saving the whole drawing as a PICT, EPS, GIF, TIFF, PDF or PNG file.

When using the clipboard or drag-and-drop for exporting pictures, pro Fit generates PICT and PDF data. PICT data is compatible with nearly all MacOS applications but it often does not yield optimum results when rendering a picture on screen or for printing. PDF data provides much better quality and portability but is not accepted by all applications. Some applications, such as Keynote version 1, accept PDF via clipboard but not via drag and drop, i.e. you should use copy and paste for placing pro Fit graphics into Keynote.

In pro Fit's preferences dialog, which you can bring up by choosing Preferences... from the pro Fit menu, there is a section titled "Copy". In this section, you can define what image data format (PICT and/or PDF) pro Fit is to generate when you copy items form a drawing menu. You can also define the depth and resolution of the PICT data to be exported.

Note: Starting from pro Fit 6.1, PICT data is exported as bitmap only because pro Fit does not natively use Quickdraw anymore.

Exporting file formats

To save a drawing as a picture to be exported to other programs choose **Export...** from the File menu and select the desired file format from the Format popup.
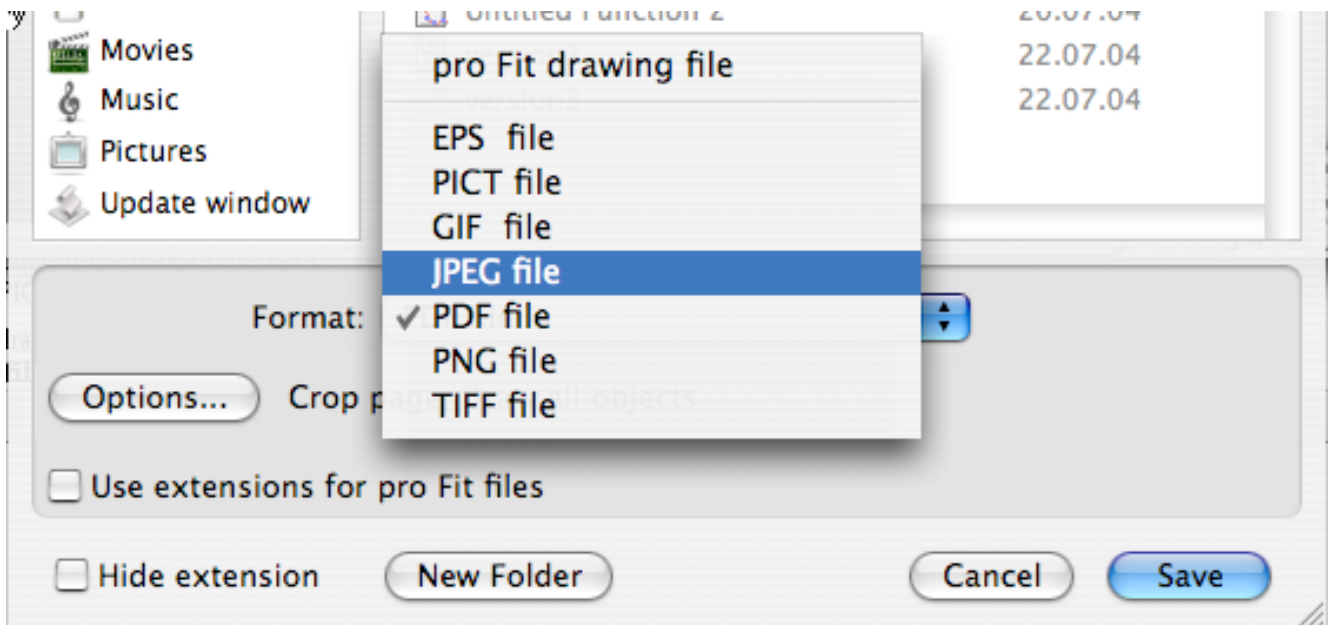
Depending on the file type you choose, you can specify various formatting options by hitting the Options button.

PDF files

Under MacOS X, PDF is the preferred graphics interchange format. It contains substantially resolution-independent vector information that will provide good results when rendered on screen or printed.

You can set the following PDF specific options by hitting the Options button:
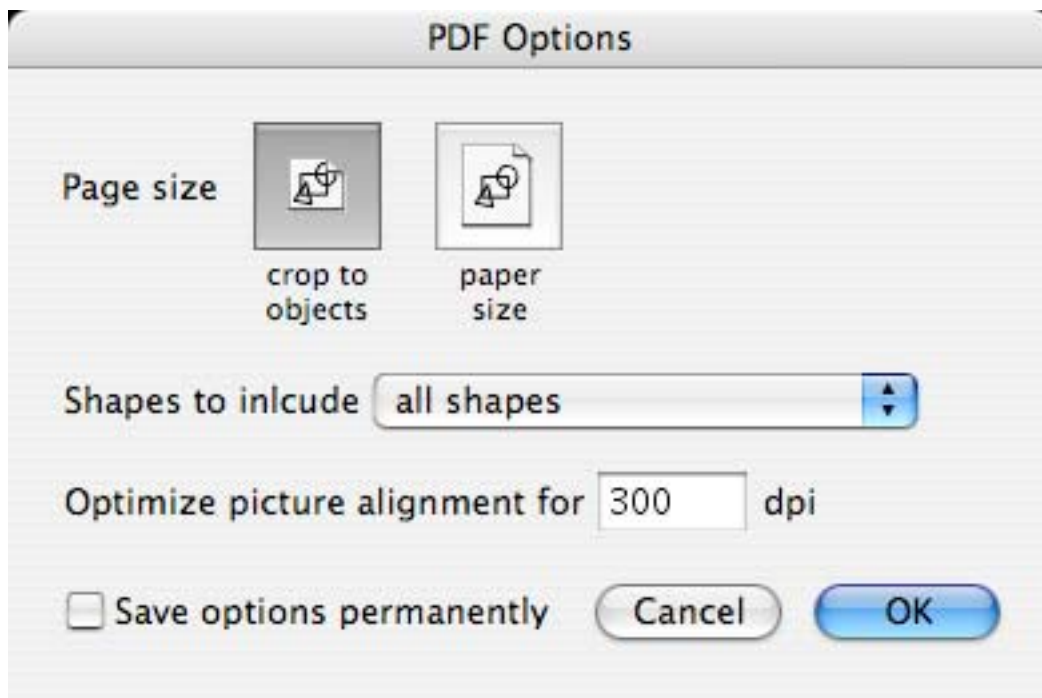
Under **Page size**, you select if the bounds of the image are to correspond to the bounds of the shapes included in the image, or if the paper size of the current drawing window is to be used. Under **Shapes**

to include, you specify if all shapes or only the currently selected shapes are to be included in the pdf file.

The option **Optimize picture alignment for** allows you to fine-tine the position of any bitmapped pictures that you may have in the graph. Pro Fit will endeavor to place the pixels of the bitmap in alignment with the pixels of a destination medium having the given resolution. Use this option if you find the pictures in the pdf file to look blurred and un-sharp. Set the value to 0 for suppressing picture alignment.

EPS files

To save a drawing as an Encapsulated PostScript (EPS) File, choose **EPS file** from the format menu of the file saving dialog box.

You can set the following EPS specific options by hitting the Options button:
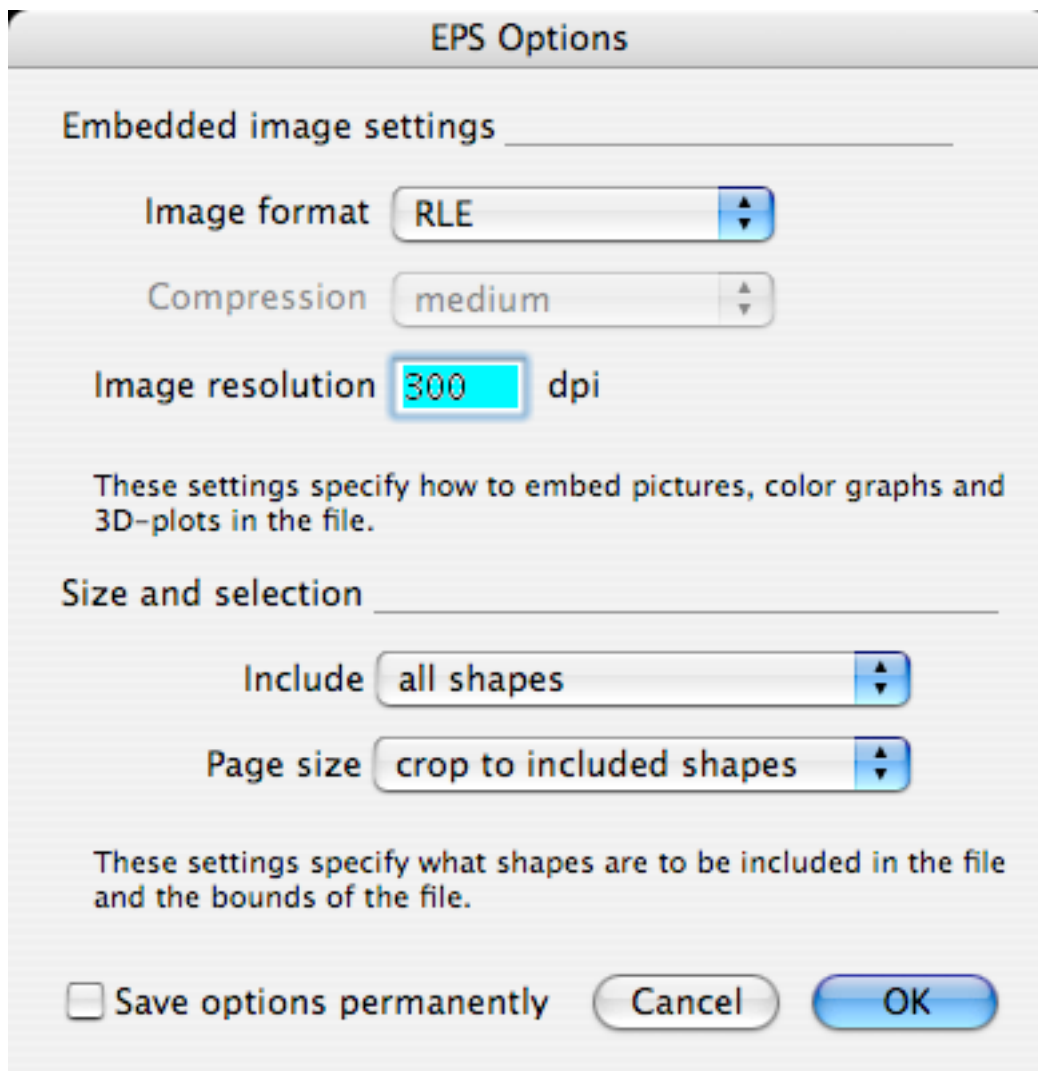
The popup **Image format** lets you specify the encoding to be used for images embedded in the drawing file. "Embedded images" in this sense are any type of graphical elements imported from other applications, as well as from some plug-ins, such as 3DplotterGL. Generally, you should use RLE encoding for line graphics because it is a lossless compression encoding optimized for this type of

graphics, while you should use JPEG encoding for photos or other images that don't have sharp edges and homogeneous areas. If you use JPEG encoding, you can specify the degree of compression using the **Compression** popup. The field **Image Resolution** specifies the resolution with which the embedded images should be stored because pro Fit will store all embedded images as bitmaps.

Use the popup **Include** to specify if all the shapes or only the currently selected shapes are to be included in the eps file.

Under **Page size**, you select if the bounds of the image are to correspond to the bounds of the shapes included in the image, or if the paper size of the current drawing window is to be used.

The size of EPS files created in this way is kept as small as possible. This small size is useful when you want to transmit your pictures over e-mail to a publisher. However, keeping a small size introduces some limitations on the number of text formatting options you can use. If a certain text-formatting option is not supported by the PostScript font you plan to use, like "Outline" or "Shadow", or "Underline", then these text formats are ignored when storing your document as an EPS file. Typographical formatting styles like Bold Face or Italic are nearly always available in all common PostScript fonts.

There is another point involved in keeping the size of EPS files small, and it is again connected to fonts. pro Fit does include information on the fonts used in your document, but does not include the fonts themselves. So make sure that you use fonts that are available to the application to which the pro Fit EPS files are imported.

A pro Fit EPS file contains a PostScript representation of the drawing for printing, and a picture to display on screen (called the *template*). The format of the picture template that is included in EPS files can be selected using the PICT options panel of the Preferences dialog box (File menu). All PICT options can be used except the "embedded PostScript" option (which will automatically be replaced by "normal"). It is advisable to use the high resolution bitmaps only if the high resolution information is really needed. Otherwise use a normal picture or a low resolution bitmap because they require less memory. PICT Options are discussed in Chapter 12, "Printing".

Note: pro Fit generates Postscript level 2 EPS files. Such files are incompatible with old devices requiring Postscript level 1, such as some of the first printers of the LaserWriter family.

> A drawing saved as an EPS file cannot be opened by pro Fit anymore. And PICT files will be opened as a single picture shape.
>
> To be able to modify it later, save a copy in the pro Fit format!

GIF, TIFF, JPEG and PNG files

To save a drawing as a Graphics Interchange Format (GIF) image, as a Joint Photographic Experts Group (JPEG) image, or as a PNG or TIFF image, choose the corresponding option from the format menu. All these file formats generate bitmap based images of your drawing. They are therefore resolution dependent, but they are compatible with a wide range of applications and platforms. For each of these formats, you can specify some formatting settings by hitting the Options dialog. The following shows an example of the settings available for GIF files, but the settings for the other bimap based interchange formats are similar:
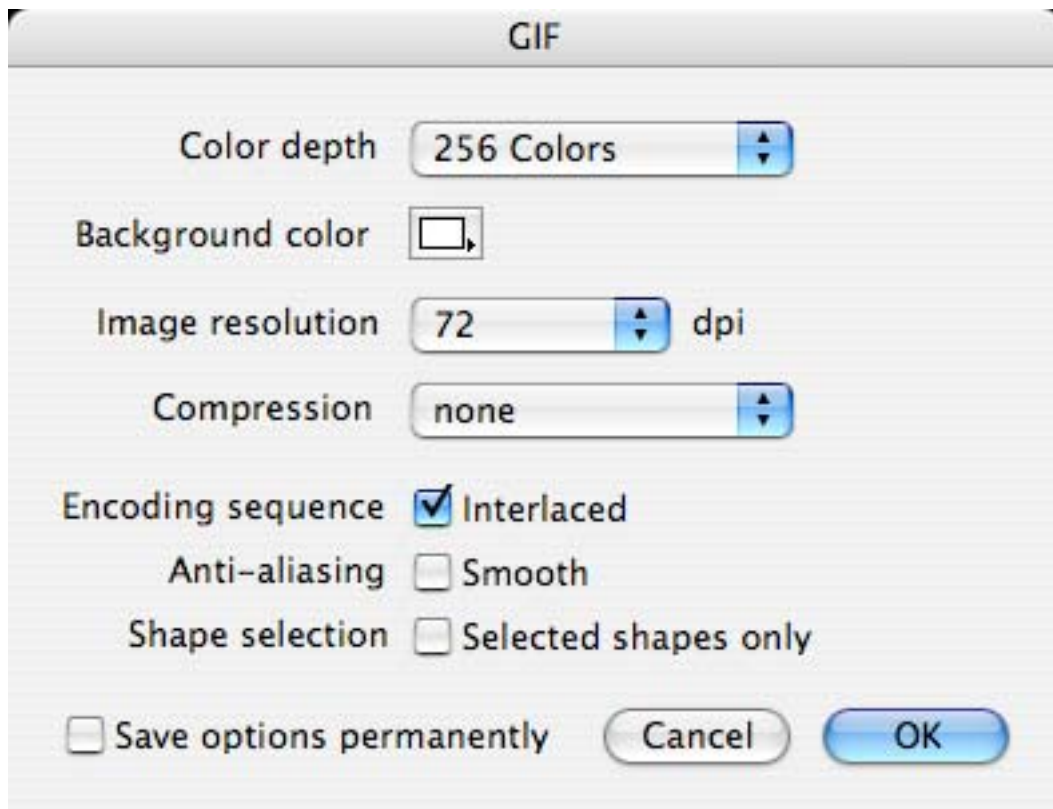
You can specify the following options:

Color depth: The maximum number of different colors in the image.

Background color: The background color of the image. In most image formats, this color is interpreted to be transparent

Resolution: Most bitmapped image formats allow you to define a resolution. Set this to 72 dpi if you inted to use the image for screen rendering only. If you inted to use it for printing, use a higher resolution. Higher resolutions will lead to larger files.

Compression: Some image formats provide compression algorithms for reducing file size. Note, however, that these compression algorithms may be lossy, in particular if a high degree of compression is used, e.g. in a jpeg file.

Encoding sequence: If this option is checked, the data is stored in such a way that the image is rendered with increasing quality when being accessed through a low bandwidth network. If this option is unchecked, the image is stored row by row, which will lead to a poor image build-up when the image is accessed through a low bandwidth network. This option is not available for all image formats.

Anti-aliasing: If you check this option, pro Fit will smooth the edges of the image by using anti-aliasing technologies. It will render a two times enlarged drawing and then reduce the resulting bitmap by a factor of 2

Shape selection: If checked, only the currently selected shapes will appear in the image. Otherwise, all shapes in the drawing window will be rendered

Check "Save options permanently" for saving the selected options as a default setting for the given image format.

Importing pictures

pro Fit can import the following image formats: PICT (Quickdraw Picture), PDF, PNG, TIFF, GIF and JPEG.

There are three ways of importing pictures: over the clipboard (by choosing **Paste** in the Edit menu), by dragging the file or image from the Finder or another application into pro Fit drawing window, or by choosing the Import command from the File menu to open a supported image file type.

Note that pro Fit imports pictures 'as a whole' and does not take them apart. If you use a drawing application to create a line and a rectangle and paste these objects together into pro Fit, they are interpreted as one picture, not as a line and a rectangle.

An imported picture can be resized or rotated, but it cannot be edited in any other way. Rotating and resizing may not work with imported pictures if they contain any non-standard information, such as PostScript commands.

To obtain size and resolution information of an imported image, double-click it. The dialog box that comes up also allows you to set or reset the size of the picture.

Plotting

pro Fit generates graphical representations of functions and data sets inside drawing objects called *graphs*.

A graph consists of two or more axes and one or more plots. Each plot represents a set of plotted data or a plotted function.

Plot types

Pro Fit supports several plot types:

Two-dimensional function plots consist of a single line. They can be used for plotting one output value of a function versus one input value. To plot a two-dimensional function plot, use the Plot Function command from the Draw menu.
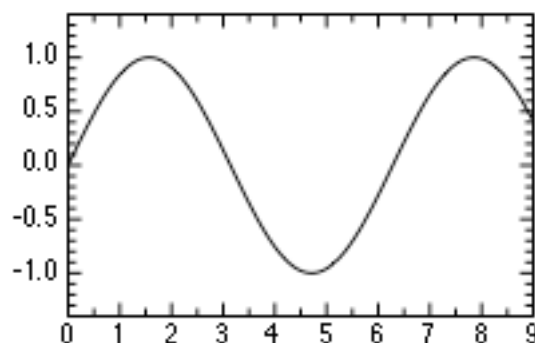
Contour plots consist of a series of contour lines and/or a color-encoded pixels representing a function's output value versus two of the function's input values or a set of data points having x-, y- and z-coordinates. To plot a function contour plot, choose one of the commands from the Contour Plot submenu of the Draw menu.
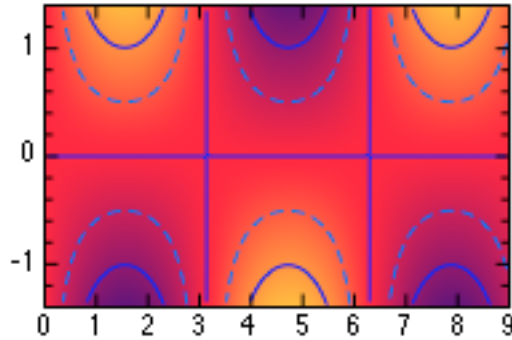
Scatter plots represent sets of data points having x- and y-coordinates. The data points can be marked by plot symbols and/or be connected by lines. To draw a scatter plot, choose Scatter Plot from the Draw menu and select the option "Scatter Plot" from the Plot Type pop-up.
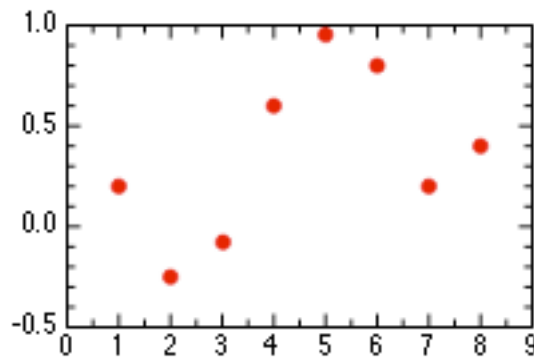
In a *bar chart*, two-dimensional data points are represented by horizontal or vertical bars. To generate a bar chart, choose Scatter Plot from the draw menu and select the option "Horizontal Bar Chart" or "Vertical Bar Chart" from the Plot Type pop-up.
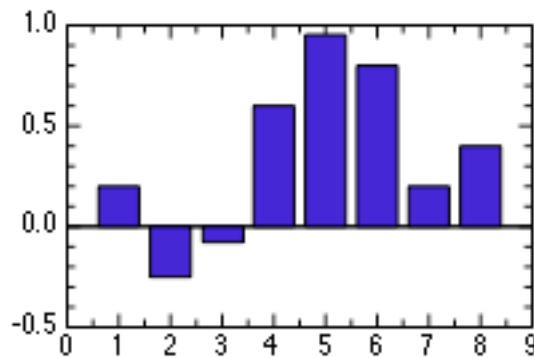
In a *skyline plot*, two-dimensional data points are represented by horizontal or vertical lines interconnected by vertical or horizontal connecting lines, giving the impression of a "skyline". To generate a skyline plot, choose Scatter Plot from the draw menu and select the option "Horizontal Skyline" or "Vertical Skyline" from the Plot Type pop-up.
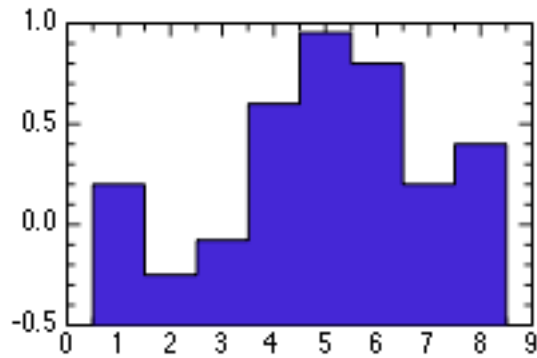
A *box plot* represents the statistical properties of one or more data set. Each data set is represented by a box, the box indicating the minimum, maximum, lower quartile, upper quartile and median of the data set. Optionally, the box can also show some or all data points in the data set. To generate a boxplot, choose Box Plot from the draw menu.
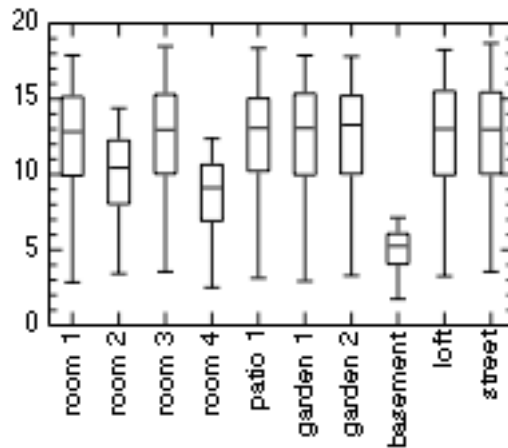
Axis types

The second important part of a graph are, besides the plots, its *axes*. A graph can have several axes.

All of pro Fit's built-in graph types have x- and y-axes. The x-axes extend, by definition, horizontally, while the y-axes extend vertically. The z-axes are not directly visible in the drawing plane. They define the scaling, color scheme and position of contour lines for z-values in color plots and contour plots.

A graph always maintains two special coordinate axes, which can never be deleted. This are the **main coordinate axes**, and are called **X1**, **Y1** and **Z1**. The other axes are called X2, X3, Y2, Y3, Z2, Z3 and so on.

The axes can have **linear**-scaling, **logarithmic**-scaling, **1/x-scaling**, or **probability**-scaling.
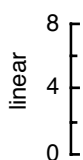
The **linear** scaling type is the standard scaling type. It indicates that there is a linear relationship between the coordinates of the graph and your paper.
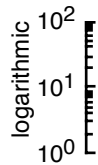
10^2 A **logarithmic** scaling indicates that there is a logarithmic relationship between the
logarithmic coordinates of the graph and your paper – it expands the lower end of an axis and
10^1 compresses its upper end. The min and max values for logarithmic axes must both be
10^0 positive.
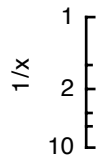
1 The **1/x** scaling type can be used to plot a function whose y-value is expected to be
1/x proportional to 1/x. If you plot such a function on a "1/x" scaled x-axis, the function is
2 a straight line. The min and max values for 1/x-axes must both have the same sign.
10
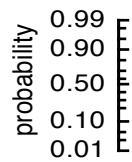
0.99 The **probability** scaling type can be used for plotting normally distributed data – or, to
0.90 be more accurate – their integral. If you have a sample of sand, and you determined the
probability 0.50 percentage of grains having a diameter smaller than x, plot this percentage as a
0.10 function of x using probability-scaling for the y-axis. If the size of the grains is
0.01 normally distributed, your data points will lie on a straight line.

With pro Fit, you can plot on any one of the coordinate axes contained in a graph, you can add new coordinate axes, and you can change their characteristics.

The next section discusses the general options that are always available when plotting. Then we discuss the procedures for plotting functions and data sets, and finally we describe how to edit and use existing graphs.

Plotting a function

To plot the output value of a function versus one of its input values:

1. **Choose the function you want to plot from the Func menu.**

2. **Set its parameters in the parameters window.**

3. **Choose Function f(x)... from the Plot menu.**

   The Plot Function dialog box appears:
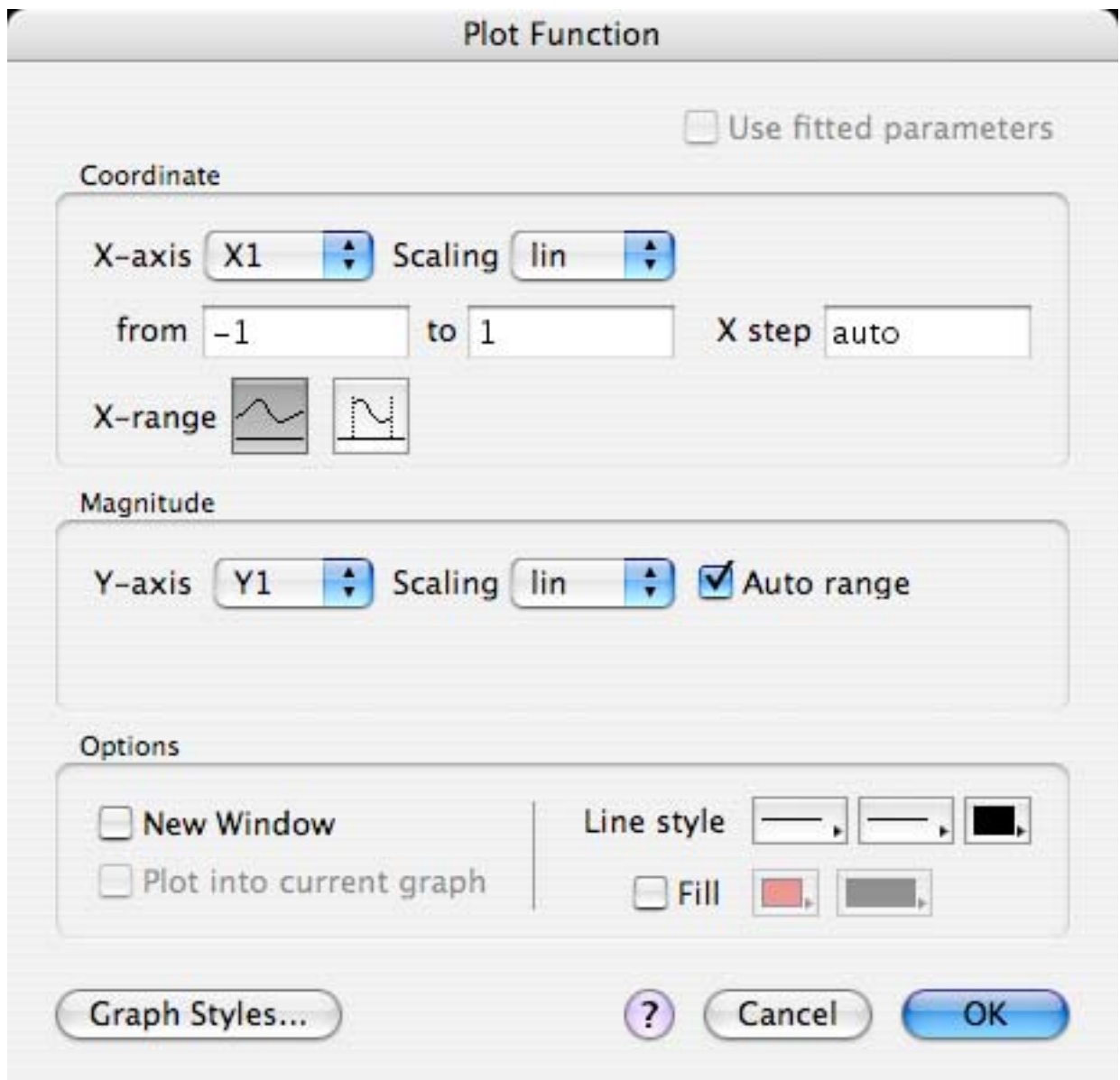
If **Use fitted parameters** is checked, the function is plotted using the parameter values calculated in the last fit. If Use fitted parameters is not checked, the parameter values in the parameters window are used.

The controls under the heading **Coordinates** define the settings for the x-axies:

- The popups titled **X-axis** and **Scaling** are used to define the x-axis of the graph to be used as well as ist scaling type.
- The edit fields titled **from** and **to** it define the range of the x-axis.
- If you are using linear x-axis scaling, the entry in the field X-**Step** determines the distance (step width) between consecutive calculated x-values. If you are using any other x-axis scaling, the field has the name **# X** and determines the number of x-values that will be calculated to plot the function. The default value for step is "**auto**". This invokes a specially designed plotting algorithm that automatically selects the x-values at which the function is calculated. If the curve representing the function is strongly bent in a given interval, then the number of points that are required for

drawing the function is large. On the other hand, if the function is a straight line, the number of points needed is smaller. The following figure illustrates this:
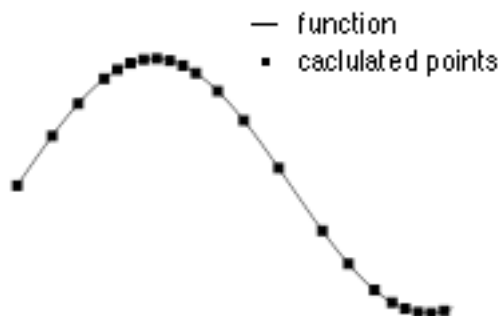
> Note that the number of calculated points is optimized for the range a function is plotted in. If you change the axes range of a graph later (e. g. for "zooming" into a detail), the number of calculated points may not be sufficient anymore for representing the curve accurately. In this case you should redraw the function to create an optimized plot for the new range.

Note that plotting with the "auto"-option results in the smallest number of data points stored to represent a function's curve. In this way you can create a plot that uses a minimum amount of memory and that is redrawn at maximum speed. However, to create such a curve, the function has to be calculated at a much larger number of points. If you are working with a slow function, you may prefer to use a fixed step to obtain faster plotting, and to go over to auto step only when you want to produce a final graph.

- The function can be plotted over the whole given range. Alternatively, you can specify start point and end point manually. Specify this by choosing one of the icons titled **X range**:
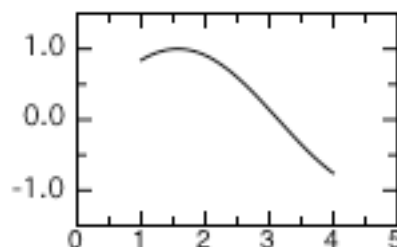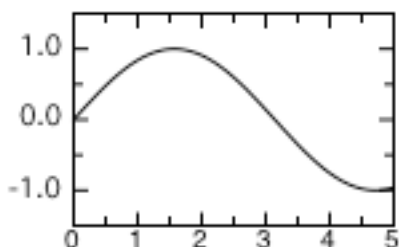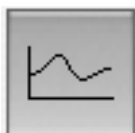
A graph with its curve from min to max (left) and a graph created using the "From.. To" option.

The two popup menus in the top left corner of the fields "X-axis" and "Y-axis" are used to choose the axis to be used for plotting, and to determine its range. The second popup menu determines the **scaling type** of the axis.

The controls under the heading **Magnitude** define the settings for the x-axies:

- The popups titled **Y-axis** and **Scaling** are used to define the x-axis of the graph to be used as well as ist scaling type.
- Check **Auto range** to let pro Fit automatically calculate the ranges of the y-axis, starting from the y-values returned by your function. If you plot into an existing graph, the ranges of the axes you use for the plot will be extended, if necessary. If you uncheck "Auto range", you can enter the ranges manually.

The controls under the heading **Options** let you choose the following options:

- Check **New window** if you want to create a new graph in a new drawing window. Uncheck this, if you want to use the frontmost drawing window.
- Check **Plot into current graph** to plot into the current graph. The current graph is usually the one where the last plotting took place. However, you can define any graph to be the current graph by double-clicking it and checking Current Graph in the dialog box that appears (Read more about this dialog box later in this chapter.). As a shortcut, you can hold down the command key and double-click the graph. If both "Plot into current graph" and "New window" are unchecked, a new graph is drawn in the frontmost drawing window.
- Use the popup **Style** to select the graph style for the new graph. A more detailed explanation of graph styles is given below.
- Use the controls titled **Line style** to specify the style of the curve representing your function.
- The controls titled **Fill** allow you to specify if and how the area below the curve is to be filled.

To plot an output value (y-value) of a function versus two of its input values, choose the command Function f(x,y) from the Plot menu:

Plot Function XYZ

☐ Use fitted parameters

Coordinates

X–axis [X1 ⬍] Scaling [lin ⬍] Input [x   ⬍]
   from [−1] to [1]   # steps [auto]

Y–axis [Y1 ⬍] Scaling [lin ⬍] Input [deg   ⬍]
   from [−1] to [1]   # steps [auto]

Magnitude

Z–axis [Z1 ⬍] Scaling [lin ⬍] ☑ Auto range

Options

☐ New Window   ☑ Draw contours
☐ Plot into current graph   ☑ Color plot of magnitude   color scheme [▮▮▮▮▶]

(Graph Styles...)   (?) (Cancel) (OK)
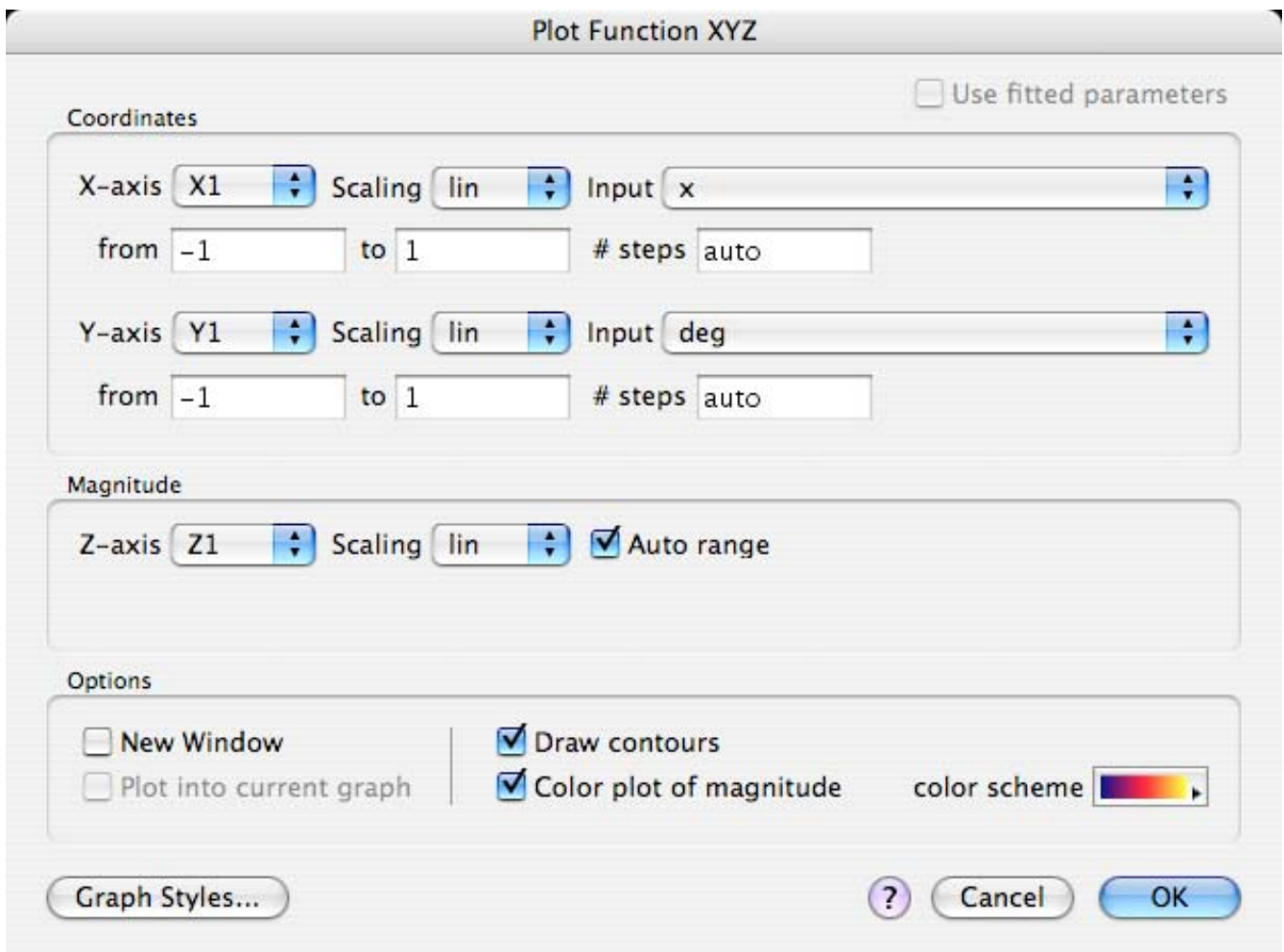
Most of the options are the same as in the previous dialog box. Note the following differences, though:

- For the x- and the y-axis, you have to specify an **Input value** .of the function corresponding to the axis. The input value can either be the function's x-value or one of its parameters.
- The z-axis defines the default color scheme and the location of the contour lines to be used, as well as the scaling to be applied to the function's output value.
- The pop-up **Color scheme** allows you to select a color encoding to use when drawing a color plot of the function.
- Check **Draw color plot** to draw a color plot, and **Draw contours** to draw the contour lines. The following two graphs show a color plot (left) and a contour plot (right) of a given function:
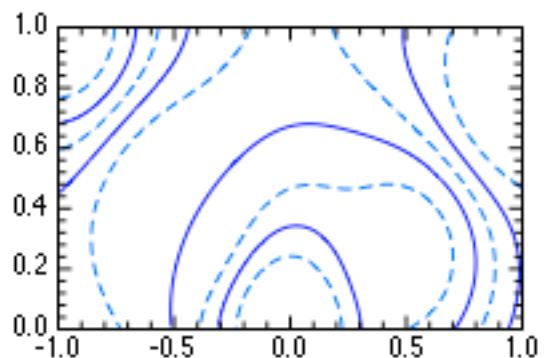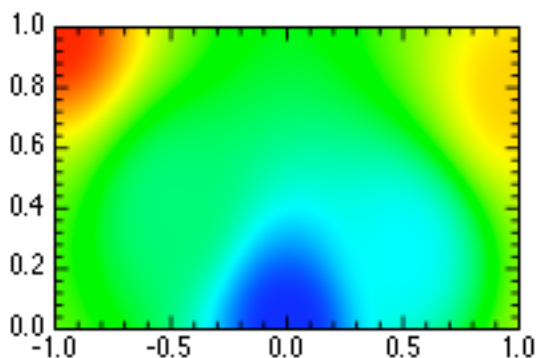
Plotting a two-dimensional data set

To create a scatter plot, skyline plot or histogram of a set of x- and y-values:

1. **Open a data window with the data you want to plot.**

2. **Choose Scatter Plot... from the Draw menu.**

The following dialog box appears:
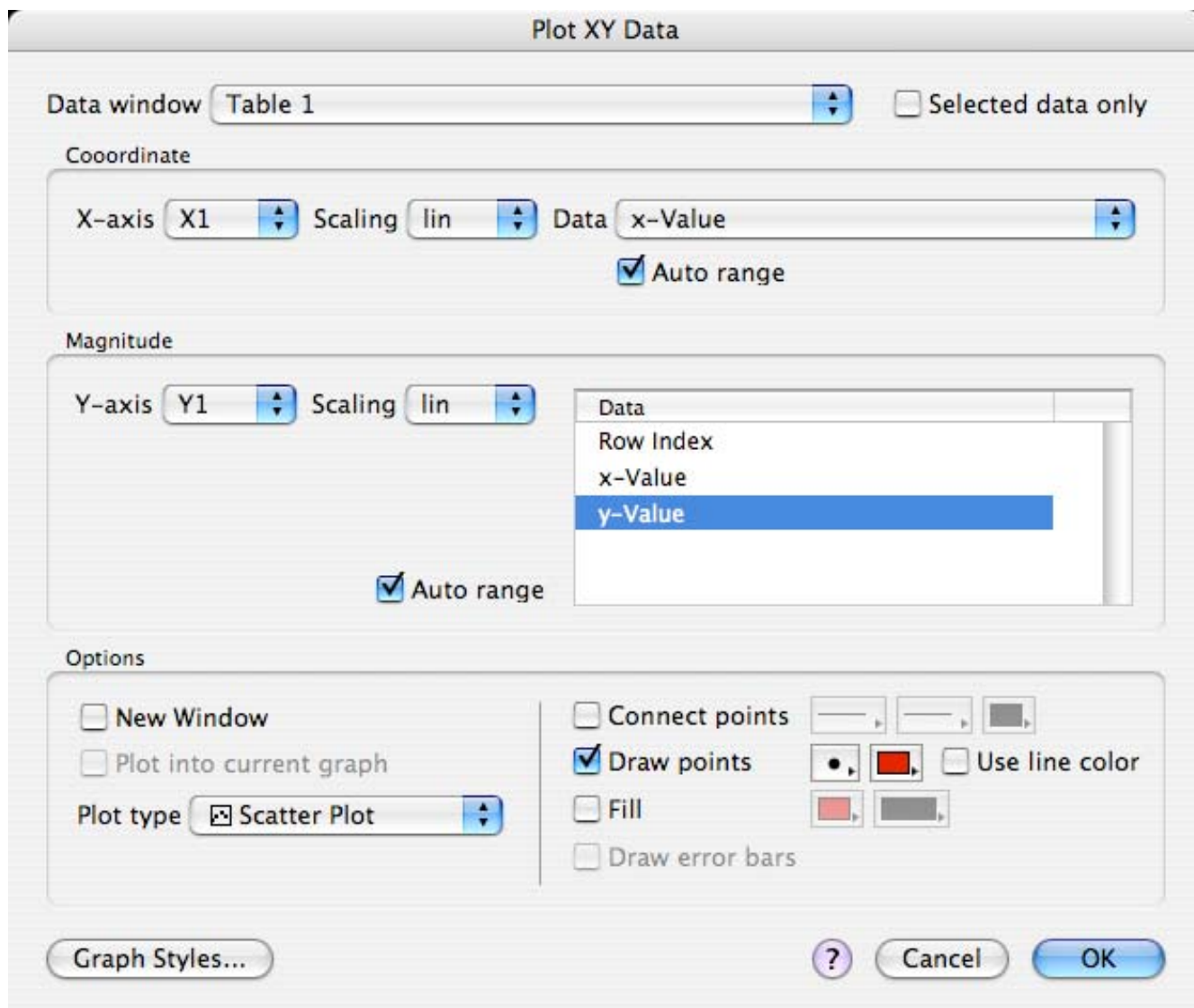
Select the **window** containing the data set, an **x-column** in the section "Coordinates" and at least one **y-column** in the section "Magnitude". Check **selected rows only** if you only want to plot those rows of the data set that are currently selected in the data window.

Then you can define the axes, their ranges, and their scaling:

- The two popup menus titled **X-axis** and **Y-axis** are used to choose the axis to be used for plotting, and to determine its range. The popups titled **Scaling** define the scaling type of the axis.
- Check **Auto range** to let proFit automatically calculate the ranges of the axis based on the values of the selected data set. If you plot into an existing graph, the ranges of the axes you use for the plot will be extended, if necessary. If you uncheck "Auto range", you can enter the ranges manually.

If you do not use **Auto range** but define your own ranges in min and max, all data points outside these ranges are ignored – only data points within the ranges of the graph are plotted and stored together with the graph. If you always want the complete data set to be stored with the graph, check Auto range and resize your graph after plotting..

The controls under the heading **Options** define the graph to be used for plotting and the plotting style:

• Check **New window** if you want to create a new graph in a new drawing window. Uncheck this, if you want to use the frontmost drawing window.

• Check **Plot into current graph** to plot into the current graph. The current graph is usually the one where the last plotting took place. However, you can define any graph to be the current graph by double-clicking it and checking Current Graph in the dialog box that appears (Read more about this dialog box later in this chapter.). As a shortcut, you can hold down the command key and double-click the graph. If both "Plot into current graph" and "New window" are unchecked, a new graph is drawn in the frontmost drawing window.

• Use the popup **Style** to select the graph style for the new graph. A more detailed explanation of graph styles is given below.

• Use the controls titled **Connect points** to specify if the data points are to be connected and, if yes, the thickness, color and dash pattern of the connecting line.

• Check **Draw points** to draw a plot symbol for each point. Check Use line color to force the color of the data points to be equal to the color of the connecting line between the data points.

• Check **Draw error bars** to draw error bars for each point using the default x-error and y-error columns of the selected data window. (To specify those columns, click into each one while holding the control key down and choose "Default X-error" or "Default Y-error". Then choose the Plot XY Data command.)

• The controls titled **Fill** allow you to specify if and how the area below the curve is to be filled.

• Use **Plot type** to select if you want a scatter plot, bar chart or skyline plot.:
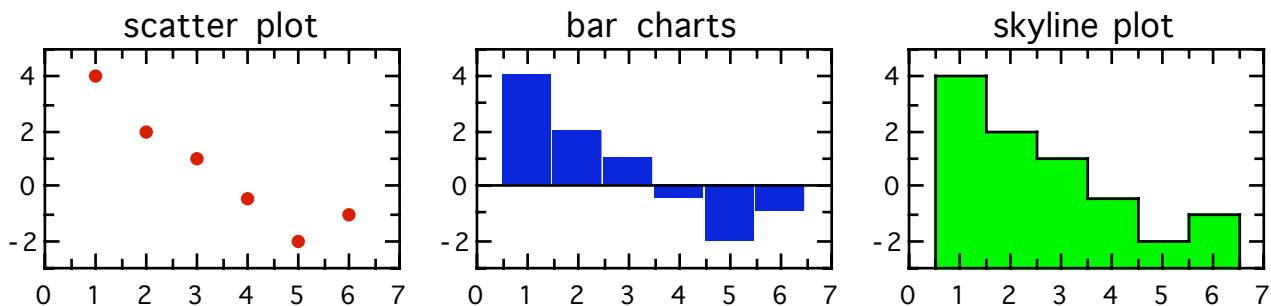
When using scatter plots, use the **Point style** pop-up menu to select a plot symbol. If you are plotting multiple data sets, only the first set will be drawn with this symbol. The symbols of subsequent sets are chosen according to the current *graph style*. See section "Styles", later in this chapter for further information about graph styles.

Bar charts and skyline plots can also start from the vertical axis:
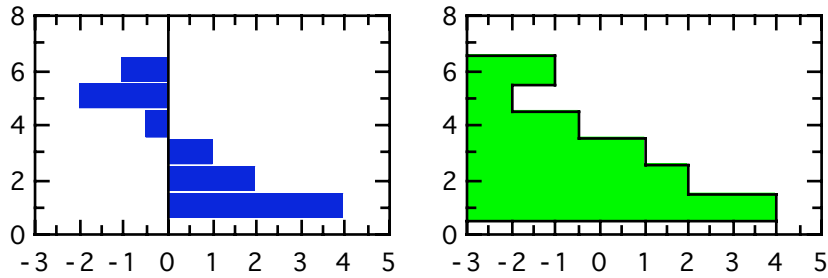
The main differences between bar charts and skyline plots are:

- In bar charts, the bars have always the same width. The width is either derived from the smallest distance of two data points or is a fixed value. In skyline plots, the "steps" can have varying width depending on the distance of the data points:
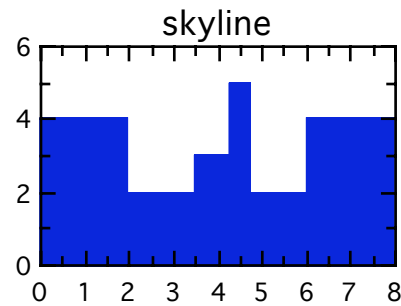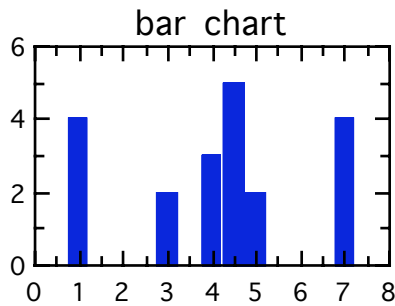
- When plotting multiple bar charts, the bars can either be behind each other or on top of each other. In skyline plots, the plots are always on top of each other.
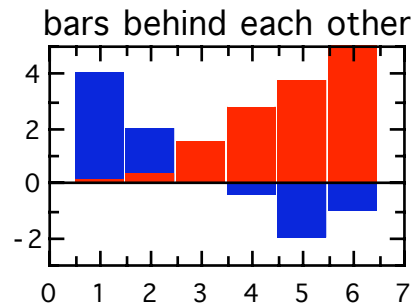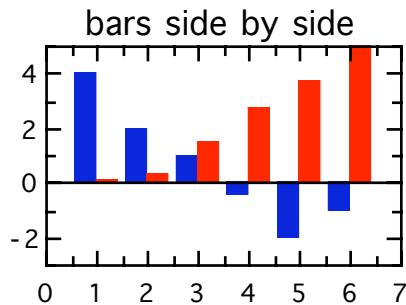
Most of the options for bar charts can be set by double-clicking the graph and selecting the "Bar charts" panel. This panel is desribed below, in the section "Bar charts panel".

Note: You can use text columns for plotting. When you e.g. create a plot using a text column as x-column, the labels of the x-axis of the graph will be set to the texts in the text column. This type of "category axes" are especially useful for histograms.

Plotting a three-dimensional data set

There are two command for plotting data sets that have z-values (in addition to explicit or implicit x- and y-values):

If your dataset is a sequence of x-, y- and z-values in three separate columns, use the command **Data z(x,y)** from the Plot menu. If your dataset is a matrix of z-values in a data window, and the x- and y-

values correspond to the row and column indices of the individual z-values, use the command **Data z(row, col)** from the Plot menu.

Both commands bring up dialog boxes where you can choose the data to be plotted. In the XYZ Columns command, you can choose the X-, Y- and Z-Column. In the Table of Z-Values command, you can choose the data window or a part thereof that holds your data.

Note: The Table of Z-Values command assumes that the rows and columns indices correspond to the current range of the X- and Y-axes of the graph. In other words, a plot created with this command will always cover the complete graph, even if you change the range and scaling of an axis.

The z-values are represented by contour lines and/or by a color encoding. The color encoding is derived from a color scheme, and its range corresponds to the range of one of the z-axes of the graph. When plotting, you can specify the z-axis to be used.

Graphs and legends

When you plot data or functions, you create a *graph object* and a *legend object*..

Graphs and legends are the most important drawing objects.
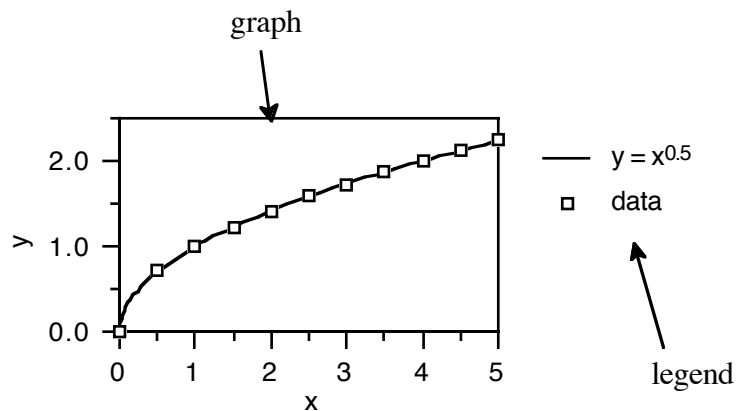
Editing legends

A legend contains a description for each curve or data set of its graph. The description consists of a symbol identifying the plot and a text. You can change the line and point style of a plot as well as the text by double-clicking the respective items in the legend.

- **Double-click** the **text** of a legend to change a the name of a plot.
- **Double-click** the **plot symbol** to choose the color, plot symbol and line styles for a plot.
  Find more information on this topic later in this chapter.

To change the space allocated for the plot symbols or the distance between lines in the legend, simply resize the legend by dragging its selection points.

A graph is logically linked to its legend and vice versa. If you change the appearance of a plot, the change is reflected in both the graph and its legend.

To mantain this relationship, only one legend can attributed to a graph. If you duplicate a legend, e.g. by option-dragging or copy and paste operations, the duplicate will be an inert picture shape not linked to any graph.

You also can ungroup a legend. This transforms the legend into a set of simple drawing

shapes, which then can be copied.

If you do not need the legend, delete it. You can always create a new legend for a graph by double-clicking the graph and checking "Draw legend" in the dialog box that appears.

Note that you can change the text style, font, and font size of a legend by selecting it and choosing an appropriate setting from the Style, Font, and Size submenu in the Misc menu.

You also can change the line styles and color of curves and lines in a legend by choosing the appropriate setting from the "Pen" and "Dash" pop-up menus in the drawing tools palette:
  • To change the line style of the first item in a legend that is drawn using a line (either connected data points or a function curve), select the legend and choose the line style in the "Pen" and/or "Dash" pop-up menus.
  • To change the line style of all items in a legend, select the legend and choose the line style in the "Pen" and/or "Dash" pop-up menus while holding down the shift key.
  • To add a connecting line to the first data point in a legend, select the legend and choose a line style from the "Pen" or "Dash" menu while holding down the option key.
  • To add a connecting line to all data points in a legend, select the legend and choose a line style the "Pen" or "Dash" menu while holding down the shift key.

By default a legend lists every plot of the related graph. You can, however, hide one or more plots from a legend by unchecking "Appears in legend" in the dialog box for editing curve styles. This is explained later in this chapter.

 A legend can be ungrouped by selecting the legend and choosing Ungroup from the Draw menu. When an legend is ungrouped, it is transformed into a set of lines, data points and texts.

Editing graphs

The possibilities for changing and editing a graph are nearly unlimited. A whole set of specialized options lets you create the graph you need. These options are accessed either by double-clicking the graph or its legend, or by using the **Graph Options** submenu in the Plot menu. (This submenu is only available if a single graph is selected or if a drawing window contains only one graph.) When you double-click a graph or choose "General..." from the Graph Options submenu, the following dialog box appears:
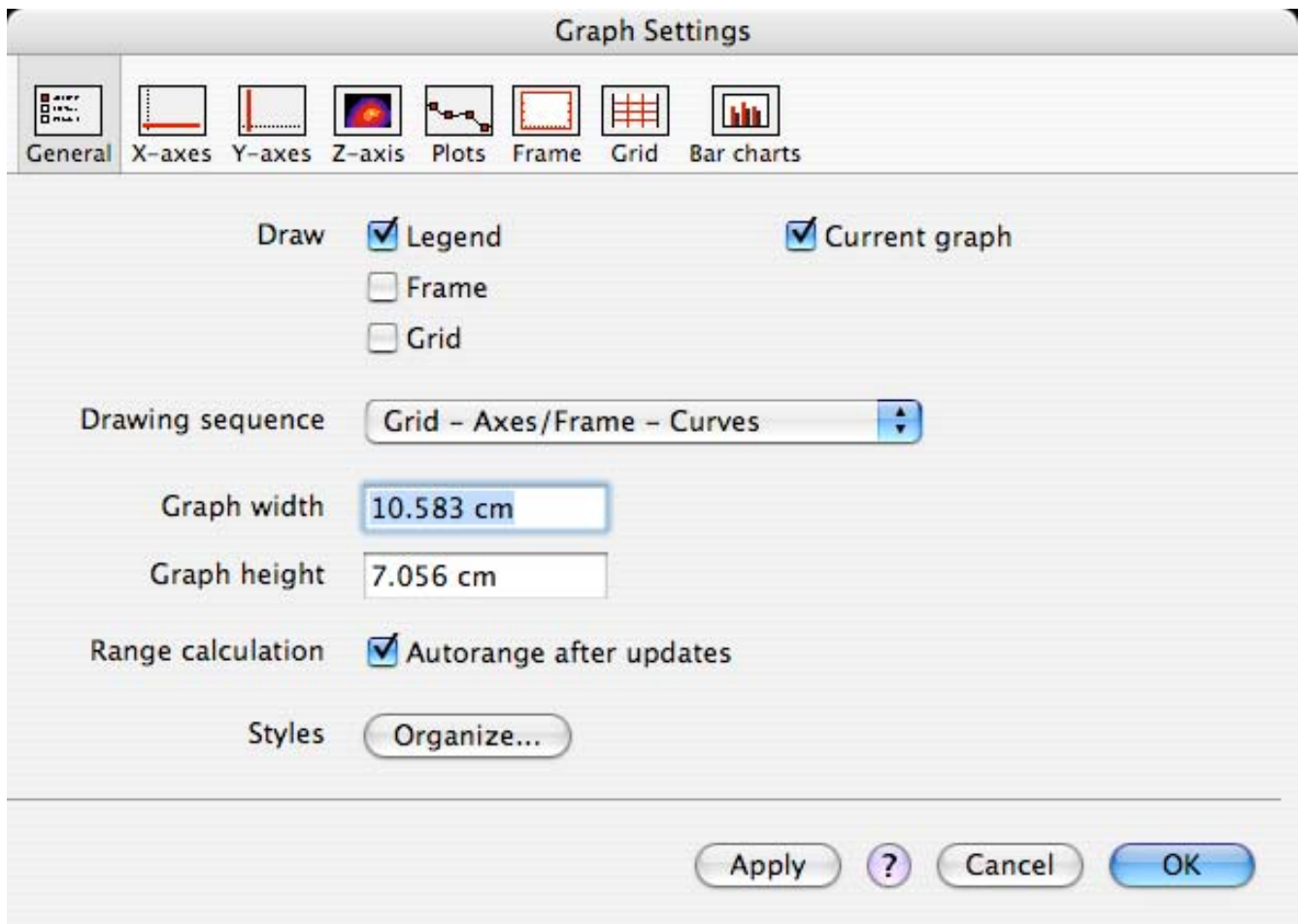
The icons ("panels") in the list at the top of this dialog box correspond to the items in the **Graph Options** submenu. Click the icons to access and edit the various parts of a graph. Click the **Apply** button to see the effects of your changes.

Panel "General"

Check **Current graph** to make this graph the currently active graph. This is the graph where plotting takes place per default.

The three **Draw** check boxes indicate if **legend**, **frame** and **grid** should be drawn or not. If you uncheck the box named legend, the legend is deleted. If you check it again the legend reappears to the right of the graph.

The **Drawing Sequence** popup menu defines the order in which the various parts of a graph (curves, axes, grid) are drawn. This is especially important if you use color to highlight your curves or if you use very large data points. A grid in front of a curve can then look quite different from a grid behind a curve.

The **Graph width** and **Graph height** edit fields let you enter precise dimensions for the graph. You can also do this by selecting the graph in the drawing window and editing its size using the Drawing Info window.

Check **Autorange after updates** to force the ranges of the graph's axes to be updated after an update command to match the updated plots. If you do not want autoranging to take place, uncheck this option.

The button **Styles** lets you save and load the current settings of a graph. A more detailed description of graph styles is given at the end of this chapter.

In the following sections we discuss the various parts of a graph and how to edit them.

Panel "Axes"

When you want to edit an axis, double click it. Alternatively, you can choose Axes... from the Graph submenu in the Menu Draw, or you can reach the axis editing panel using the list of icons in the Graph Settings dialog box.

The axis editing panel for x-axes looks like this:
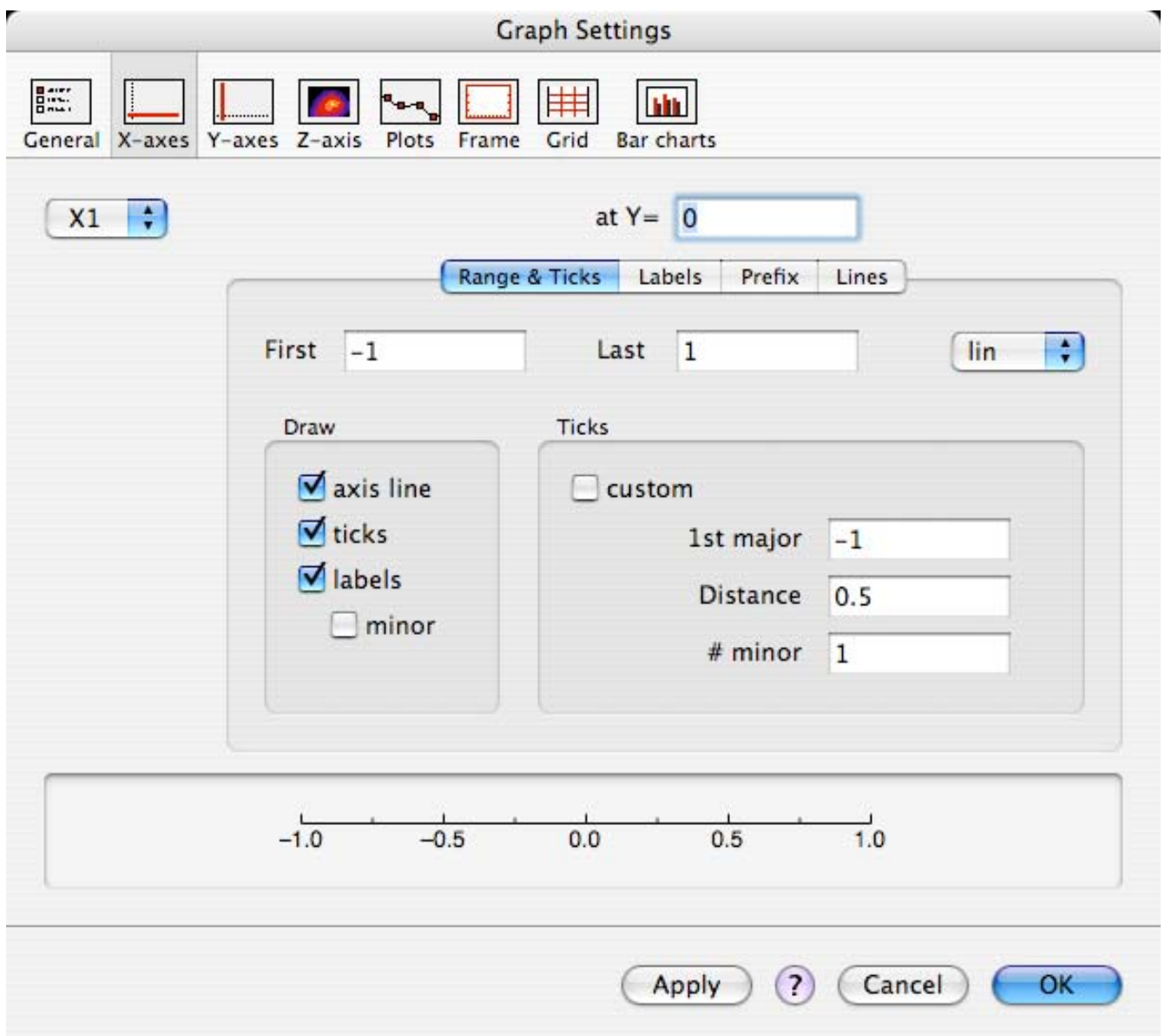
Use the popup menu in the top left corner to navigate between the various axes, to create a new axis, or to delete the current axis. (The X1 and the Y1 axes are the **main axes** and cannot be deleted.).

The edit field in the top right corner gives the position of the selected axes in the main axes coordinate system. Use this field to change the position of a horizontal (or vertical) axis with respect to the vertical (horizontal) main axis coordinates. The position is set by default to the minimum and maximum bounds of a plot when it is first created.
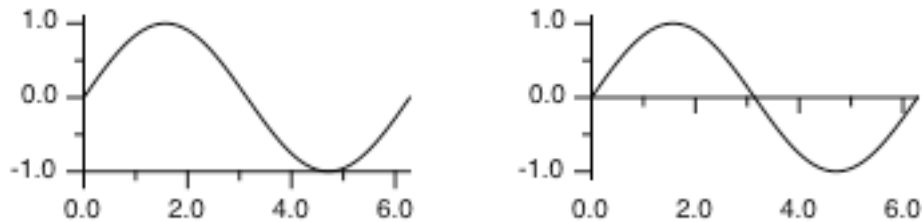
Two graphs with different vertical positions of the horizontal axis.

If the dialog box does not show the main axis (X1 and Y1 are the main axes) an additional check box is present. It is called **Same as X1** (or Same as Y1).
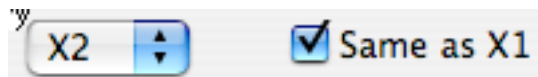
If **Same as X1** is checked, most settings of the selected axis (such as the range, scaling, color, line thickness, tick positions) are taken from the main X1 axis.

> If you want to use two different axes for the top and for the bottom of your plot, you have to uncheck this box before making any changes.

The tabs buttons **Range & Ticks**, **Labels**, **Prefix** and **Lines** let you switch between different sub-panels that are used to edit the general appearance of an axis, the appearance of its labels, and the kind of lines that are used to draw the axis and its tick marks.

If you check General, you can set the following options:
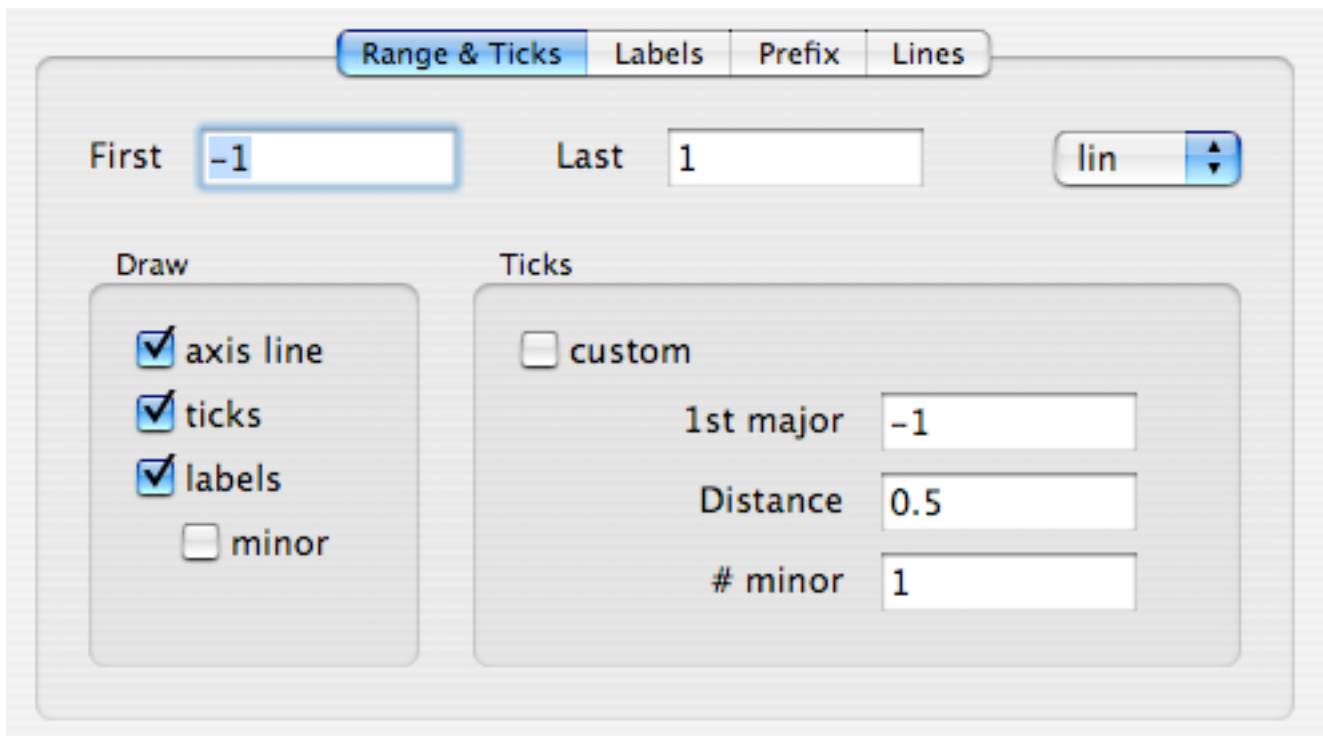
The **Draw** check boxes determine which parts of an axis are drawn.

The **First**, **Last** fields and the popup menu to their right are used to edit the range of the axis and its scaling type. See the beginning of this section for a discussion of scaling types. Note that First can be larger than Last if you want to reverse the axis.

The **Ticks** field to the right of the Draw check boxes is used to edit the tick marks. Enter the first major tick, the distance between major ticks, and the number of minor ticks between two consecutive major ones.

The edit field **1st major** gives the coordinate of the first major tick on the axis.

For a linear axis the **Distance** field defines the distance between the major ticks. For a logarithmic axis this field changes its name to **Decades** and defines the number of decades between major ticks. For a 1/x-scaling the edit fields work in the same way as for linear scaling. For probability scaling, you can edit the list of tick marks directly using the **Custom** check box.

For a linear axis the **# minor** field gives the number of minor ticks that are drawn between two major ones. For a logarithmic axis this field is replaced by a check box called **small ticks**, which must be checked to draw the minor ticks. If major ticks are drawn for each decade, the minor ticks are drawn for each multiple of ten. If there is more than one decade between major ticks, the minor ticks are drawn at all the powers of ten between the positions of the major ticks.

Instead of automatically calculating the positions of individual ticks, you can set them manually. Check the **custom** check box. This changes the contents of the ticks field.

A list appears that contains all the ticks of the axis. To add a tick, click a free space in the list (there is always a free space at the bottom of the list) and enter the desired coordinate.
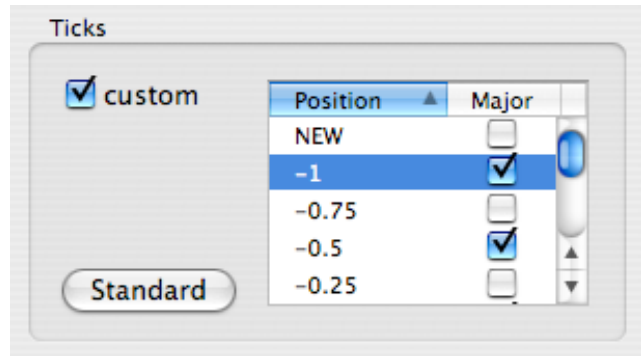
To remove a tick, select it in the list and press the delete key. To change the position of a tick, click it and enter a new value. Check **major** to create a major tick. Major ticks are written in **bold face** in the ticks list. Click the **Standard** button to automatically re-calculate the tick positions according to the present axis settings.

To set the label of a tick mark to some general text instead of a number, **double-click** the label in the drawing window. The text edit dialog box appears and you can then enter any kind of text you want.

Click **Labels** in the axis dialog box to edit the format of the labels. The inner part of the dialog box now looks like this:
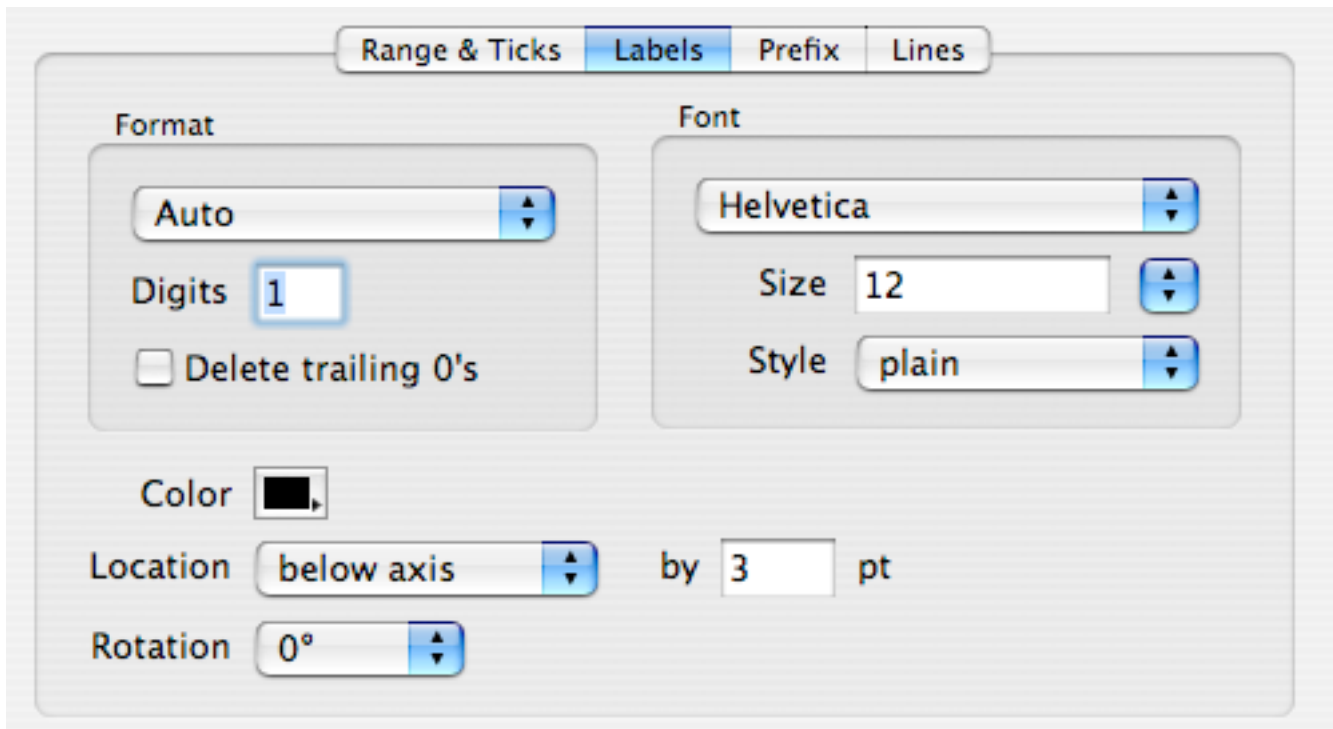
Use the **Format** field to set the format of the numbers. Choose **Decimal** to suppress exponential representation, **Auto exponent** to have all labels in exponential format with varying exponent, **Fixed exponent** to have all labels in exponential format with a common exponent.

The **Digits** field defines the number of digits to be shown after the decimal point. Check **Delete trailing 0's** to cut off any trailing 0 digits after the decimal point, which is particularly useful for logarithmic axes.

Use the **Font** field to specify the text font, size, and style to be used for the labels of the current axis.

The **Location** popup menu defines where the labels of an axis are drawn. The edit field to its right defines the distance between the labels and the axis or the frame of the graph The value in this field is in points (= 1/72 inch or 0.35 mm). Note that it can also be negative.

The pop-up titled **Rotation** allows to to rotate the labels by multiples of 90°.

Click **Lines** to change the appearance of the lines used for drawing the axis and its tick marks, and to set the position where the tick marks are drawn. The inner part of the dialog box now looks like this:
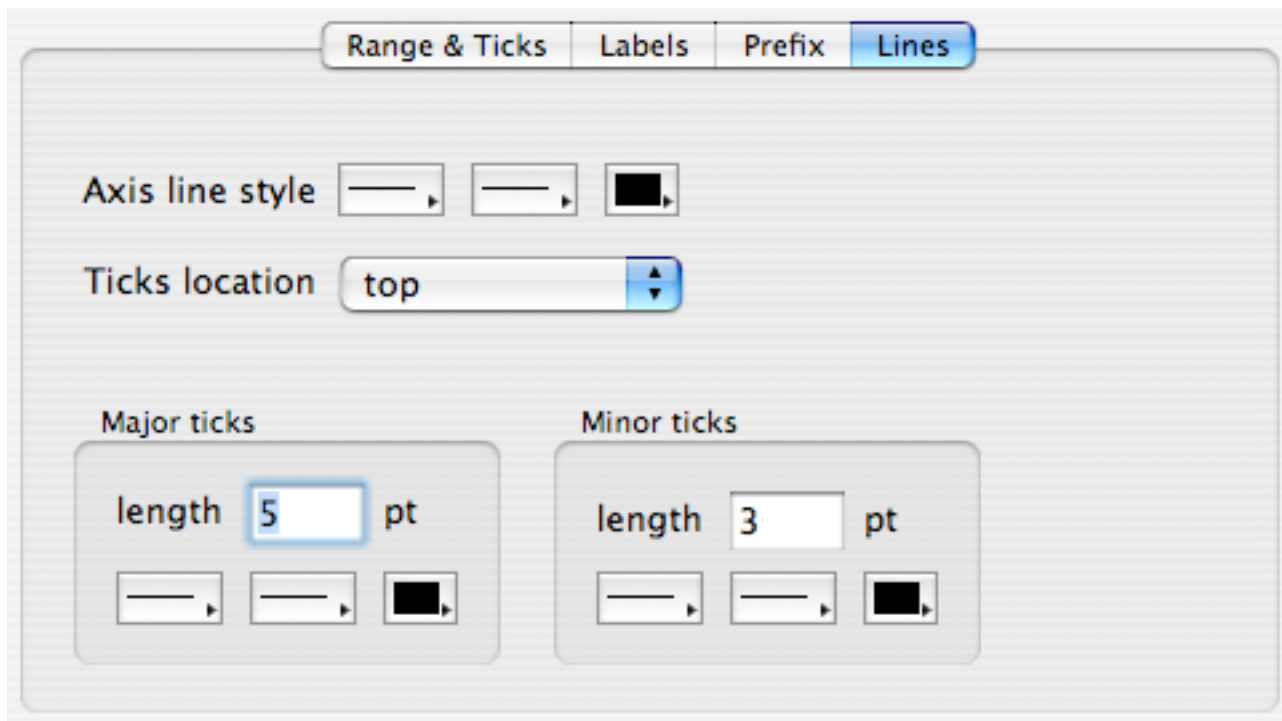
Use the **Ticks location** popup menu to set the position of the tick marks. In the **Major ticks** and **Minor ticks** fields you can set the line style, length and color of major and minor tick marks. The line style used to draw the axis can be edited using the "Axis line style" popup menus.All the options outlined above for editing axes let you create many different kinds of graphs. Note that you can create new axes and change their scaling, tick marks, etc., also if you don't use them to plot any curve.

For example, you can uncheck the "Same as X1" check box in the X2 axis panel and edit it to reflect a completely different scaling, labels style, and range than the X1-axis.

A typical application for this is a graph that displays its x-values on its horizontal bottom axis and the reciprocal x-values on its top axis.
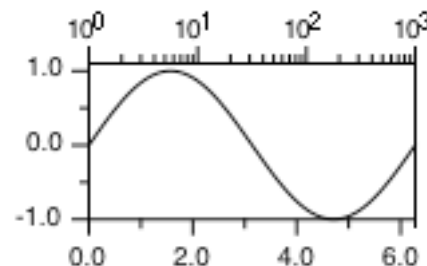
A graph with a different coordinate axis as the "X2" axis..

As an example, imagine that you have a set of data that was measured for different light wavelengths between 400 and 1000 nm. You would like to plot your data as a function of wavelength, but you

would also like to have a reading for the light energy in eV on the top axis. The energy of the light is inversely proportional to the wavelength, so you have to use 1/x scaling for the top axis.

To create such a graph:

1. **Create a graph with an x-axis from 400 to 1000.**

   Simply plot your data between these limits. Choose **Scatter Plot...** from the Draw menu. Make sure that you create a new graph by unchecking 'Plot into current graph' in the dialog box that comes up.

2. **Double-click the upper x-axis ("X2"-axis).**

   The axis dialog box (see above) for the top axis appears.

3. **Uncheck "Same as X1".**

   Do this to make sure you only change the top x-axis, leaving the bottom x-axis alone.

4. **Change the axis scaling to "1/x".**

   Be sure that the "General" radio button is selected and use the scaling popup menu, found to the right of the edit fields for the axes ranges.

5. **Enter 1.2398 for First and 3.0996 for Last.**

   A wavelength of 400 nm corresponds to an energy of 3.0996 eV, while a wavelength of 1000 nm corresponds to an energy of 1.2398 eV.

6. **Edit the tick marks**

   Do this by changing the values in the Ticks field. Note how the density of ticks tends to increase for larger values. Go over to custom ticks and edit the ticks list directly if necessary.

   The end result could be something like this:
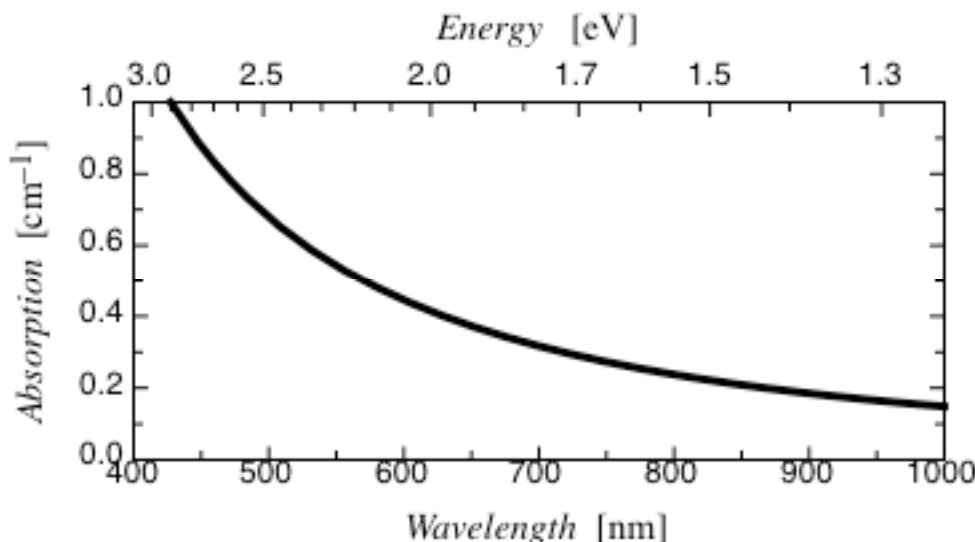
Note that the top axis, which has 1/x scaling, has the smallest value to its right and the largest value to its left.

Click **Prefix** in the axis dialog box to set pre- and postfix for the labels, to multiply them with a given factor or to offset them by a given value:
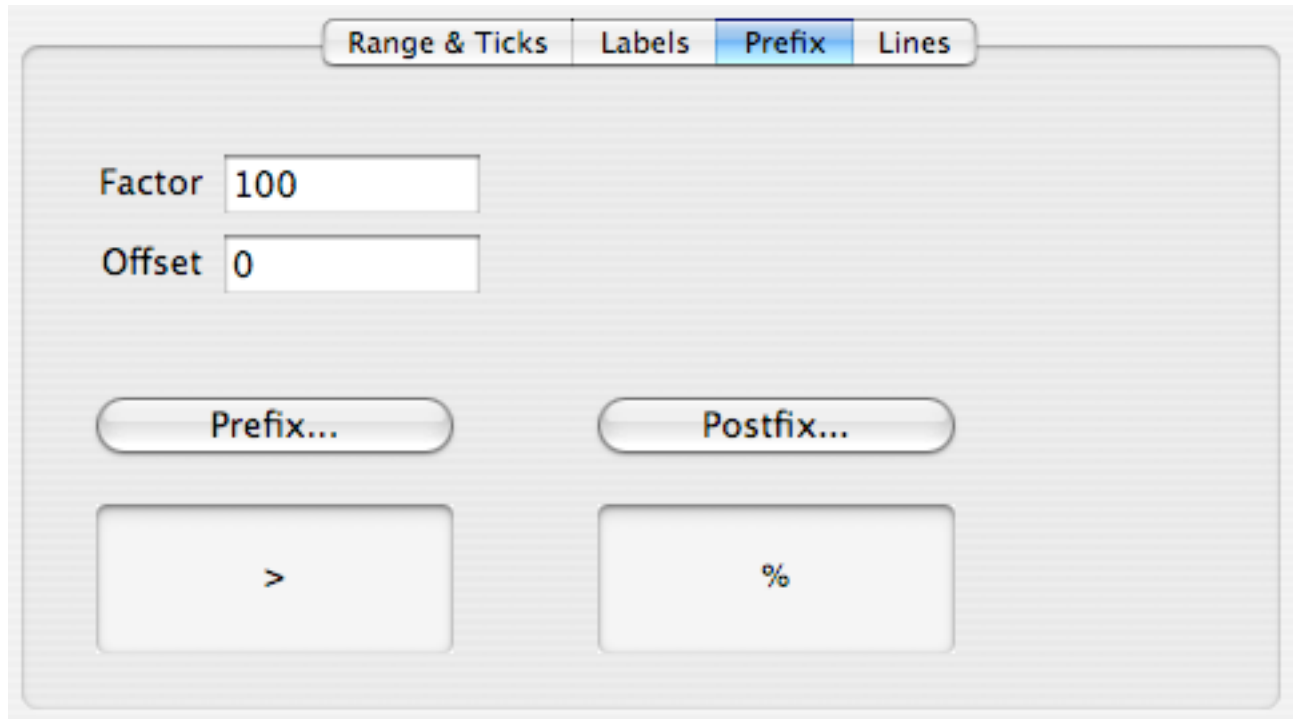
Click **Prefix** or **Postfix** to prepend or append a string to each label.

The value in field **Factor** is multiplied with the value of each label before its string is generated. You may e.g. enter 100 here to display values between 0 and 1 in percent.

The vaue in the field **Offset** is added before the string of the label is generated.

Panel "Plots"

You can change the appearance of the individual plots in a graph in many ways. Choose **Plots** from the Graph submenu (Draw menu) or click the Plots icon in the Graph Settings box. You can also double click a plot symbol in the legend.

The Graph Settings dialog box now displays the curves editing panel.
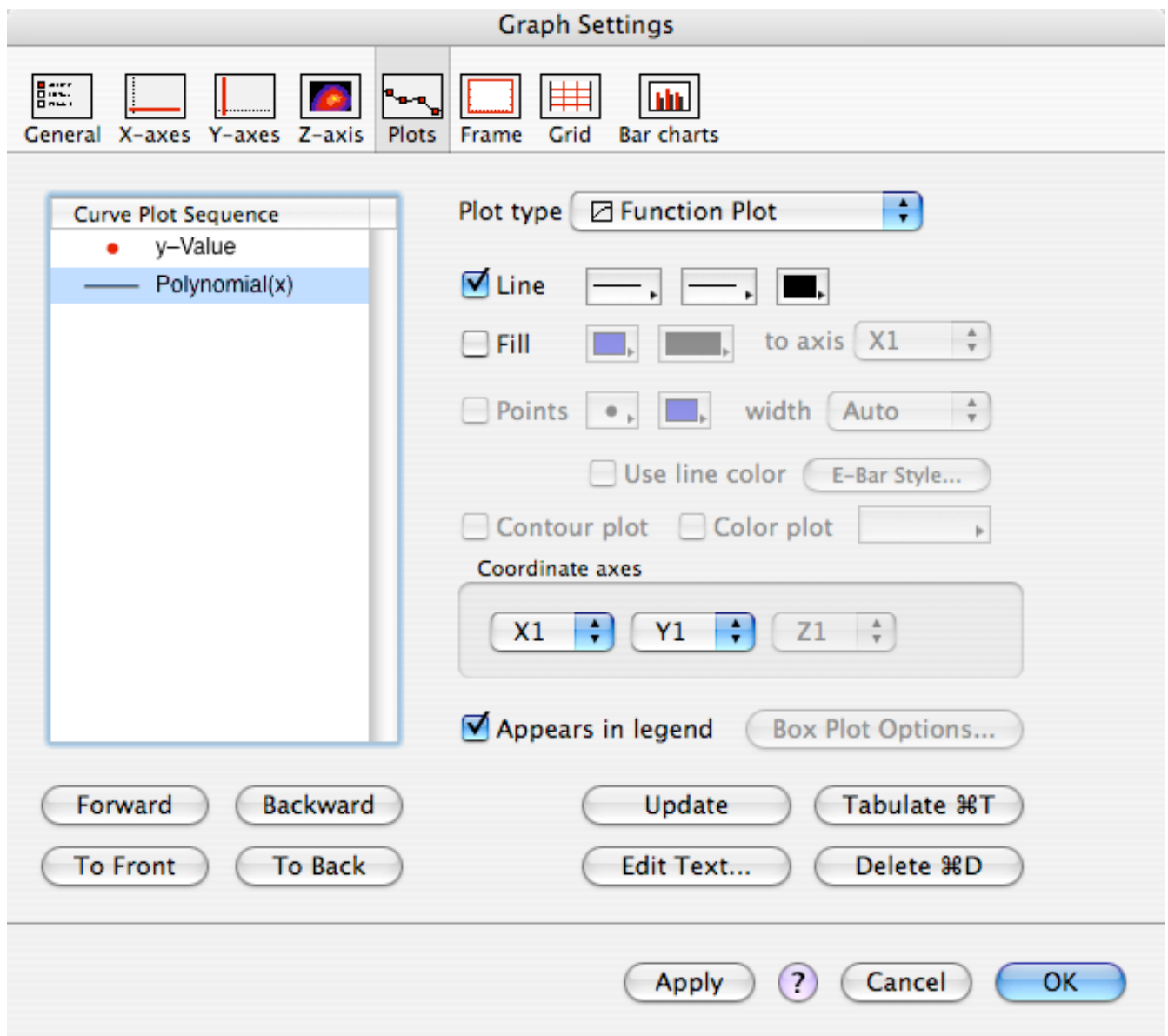
Here you can select and change or delete all curves and data sets of a graph.

To change the **drawing order** of the plots, select a plot (by clicking it in the list) and click **Forward**, **To Front, Backward** or **To Back** to move it one position backward or forward or to move it to the back or front of all plots. The first plot symbol at the top of the list is drawn first, so back means top of the list, and front means bottom of the list.

To change the **text** describing a curve or a data set, select the curve in the list. and click **Edit Text...**. Instead of doing this you can also double-click the item in the list of curves and data sets.

To specify if a data set is to be show as scatter plot, bar chart or skyline plot, use the menu **Plot type**.

The pop-up menus titled **Line** let you edit the line that draws a curve or connects the data points.

The pop-up menu **Points** lets you select the symbol for data points. Check **Line** to draw lines between successive data points or for a skyline plot. The menu **Thick** defines the line thickness used to draw the data point symbols. It can be set to **auto**, in which case the line thickness will be chosen depending on the size of the data points.

Click the **E-Bar Style...** button to define the style of the error bars.

You can fill the region between a curve and one of the axes with a color and pattern of your choice. To do this, check **Fill** and select the axis towards which the plot must be filled and the fill color and pattern using the popup menus to its right.

Check **Contour plot** to draw contour lines for the z-values of a three dimensional plot. To define the location of the contour lines, change the settings of the z-axis attributed to the plot.

Check **Color plot** and select a color scheme to draw a color-encoded image of the z-values of a three dimensional plot.

The **Coordinate Axes** popup menus define the coordinate axes used by the selected curve or data set. With these pop-ups menu you can change the reference axis of any given curve.

> Doing this for function curves which were drawn with **auto step** is not recommended. If the scaling of the original axis and the one of the destination axis differ considerably, the results can be disappointing. Remember that a function curve is only defined by a set of points. pro Fit calculated these points in an optimized way when it plotted the function *for the axis scaling and range on which the function was plotted*. If you then change scaling or range, your curve may loose its smoothness. In such a case it is better to redraw the function curve on the new axis

Check **Appears in legend** to make the curve or data set appear as an entry in the legend. Uncheck this check box to hide the corresponding entry in the legend. When an entry is visible in the legend, you will usually change its style by double-clicking it. When an entry is not visible in the legend, you must choose Curves... from the Graph submenu to access and change the style of the corresponding curve or data points.

Click **Tabulate** to recover the original data points that were used to draw the plot. In this way you can retrieve data points from a drawing when you have lost the original data set, or you can obtain a list of the data points that pro Fit calculated to draw a particular function.

Click **Delete** to delete the curve or data points from the graph. You can use the delete (backspace) key as a keyboard equivalent for this button.

Click **Update** to force a function plot to be redrawn with the currently selected function using the parameters displayed in the parameters window. For data plots, the name of the button changes to **Data/Errors…** and clicking it allows you to select the data set linked to the plot and to add or change the error values.

Panel "Frame"

A frame is a rectangular box around your graph:
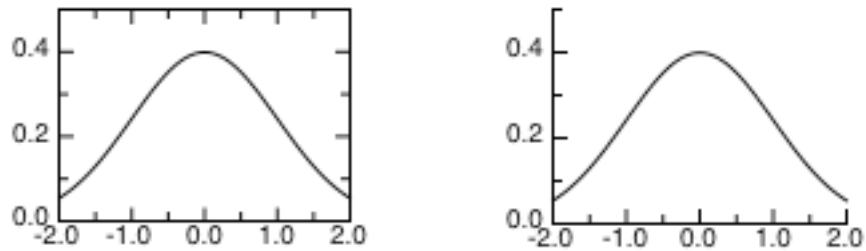
An unframed and a framed graph.

To change the appearance of a frame, either double click a graph and click the **Frame** icon, or choose **Frame** from the submenu Graph in the Draw menu
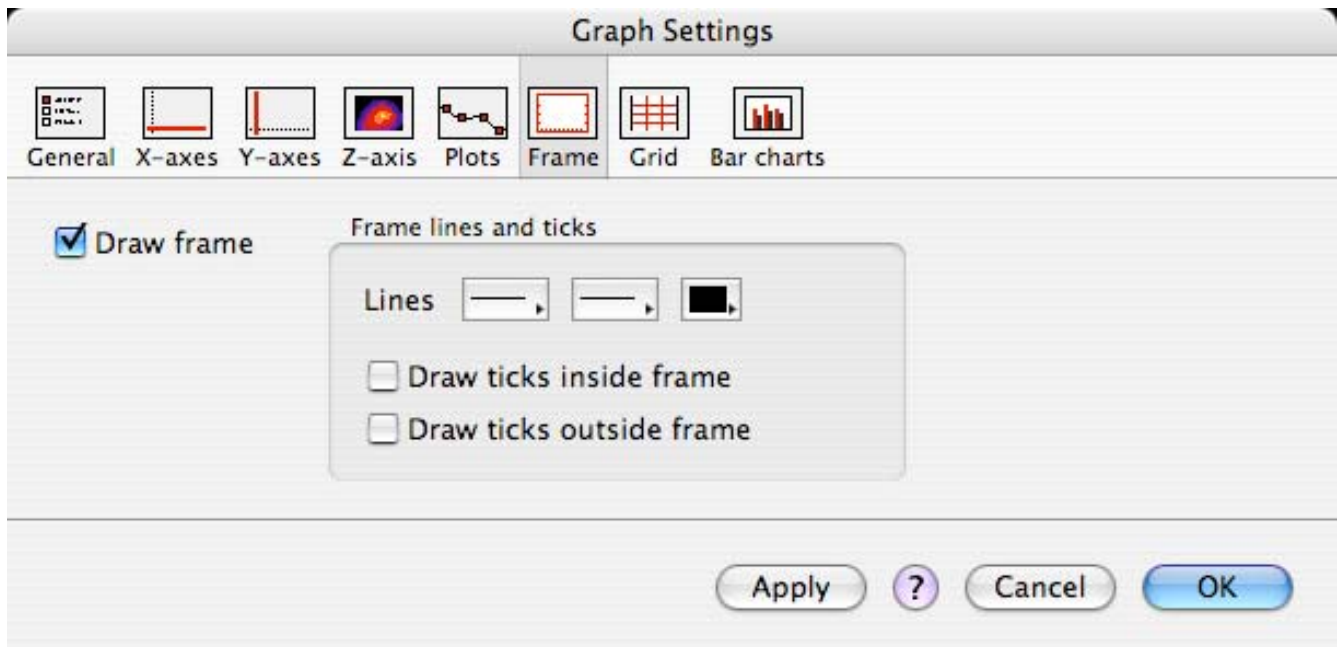
In the dialog box that appears you can edit the Line style of the frame, and determine if tick marks must be drawn on it. The tick positions of the main coordinate axes (X1 and Y1) are used. If you draw a frame with ticks, you usually do not wish to draw the axes ticks as well: Uncheck the corresponding check boxes in the axis dialog box.

Panel "Grid"

Grid lines are horizontal and vertical lines at the positions of the ticks.
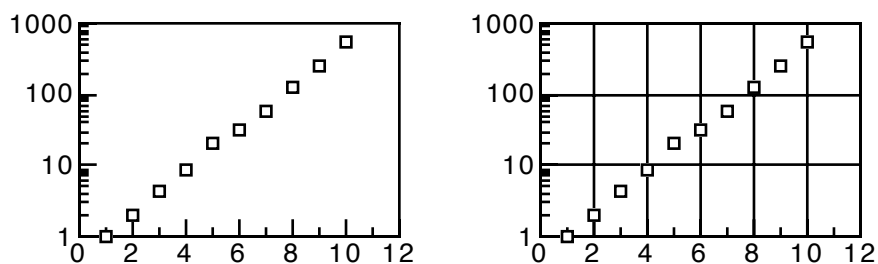
A graph without and with grid lines.

To add grid lines to your graph, double click a graph and check **Draw grid**. This will add horizontal and vertical grid lines. To customize the grid lines click the **Grid** icon in the same dialog box or choose **Grid** from the Graph submenu:
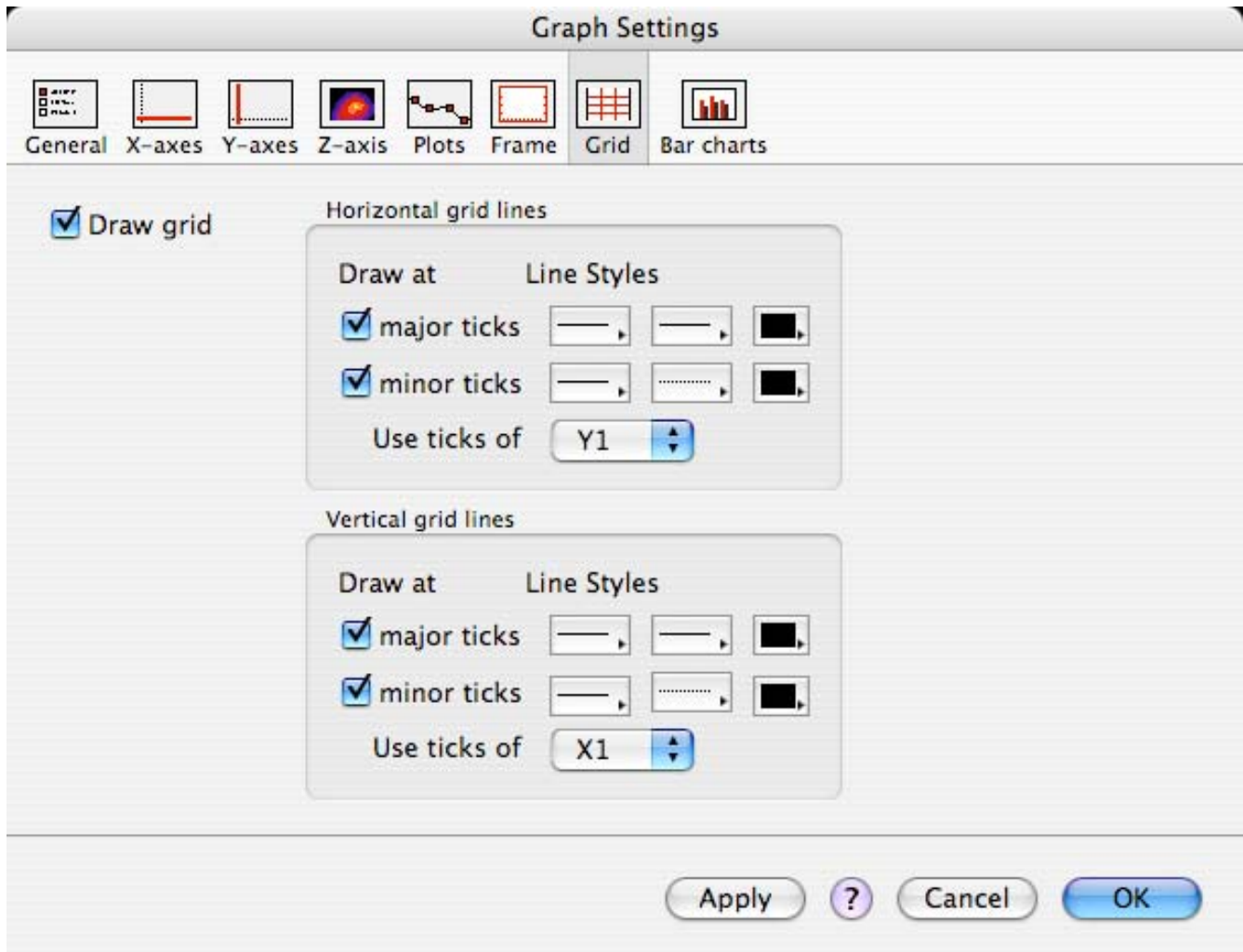
In the Grid editing panel that appears you can define where you want to have horizontal and/or vertical grid lines, and if you want to see them at minor ticks, major ticks, or both. You can also choose which axes must be used as a reference to draw the grid lines. The grid lines are drawn at the tick marks of their reference axis. By default, the ticks of the main axes (X1 and Y1) are used.

Panel "Bar charts"

To select the options for displaying bar charts, double-click the graph and select the "Bar charts" panel:
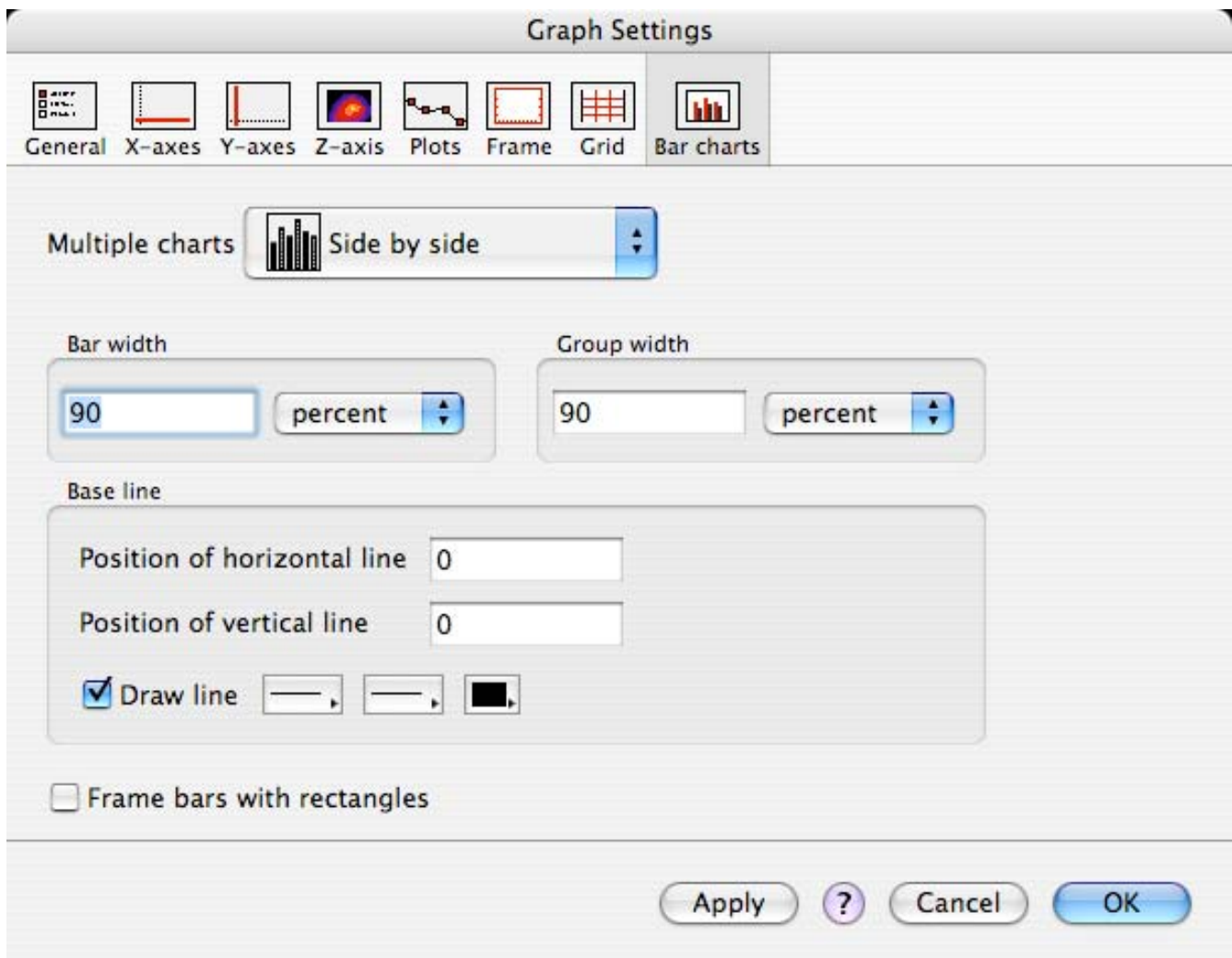
The pop-up at the top defines how to draw multiple bar charts in a single graph. They can either appear side by side or on behind each other:
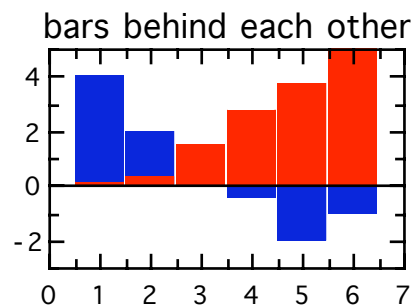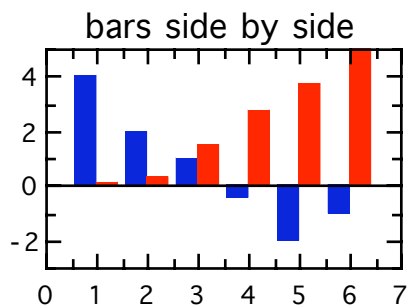
The settings under **Bar width** define the width of individual bars. The width can either be a percentage of the available space or an absolute value (in pixels, centimeters or inches).

The settings under **Group width** are used when drawing multiple bars in "side by side" mode. In this case, each group of bars (bars for the same value) has the given group width, which can again be a percentage of the available space or an absolute value (in pixels, centimeters or inches).

The **Base line** is the line the bar charts start from. There is a horizontal base line for vertical bar charts and a vertical base line for horizontal bar charts. You can set the position and line style of each base line.

Check **Frame bars with rectangles** to select how bar charts are framed:
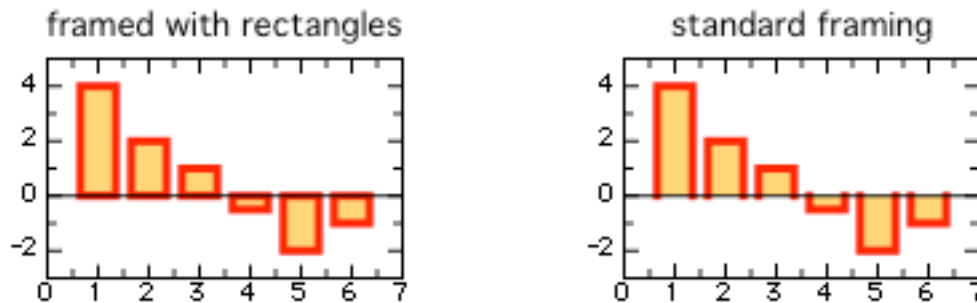
(Note: You must check the option "With line" in the Panel "Curves" for the framing to appear.)

Graph Styles

The appearance of a graph is defined by many parameters, such as its size, the ranges of its axes, the number of minor ticks, the symbols used for plotting, etc. These settings are called the *style* of a graph. You can save the style of a graph to use it (or parts of it) later for another graph. Styles are saved in the preferences file.

By using styles, you can create graphs with equal formats, e.g. graphs having the same size, the same length of the ticks, the same fonts, etc.

To save the style of a graph you can either double-click the graph and click the button Organize in the dialog box that comes up, or you can choose Styles... from the Graph submenu (in the Draw menu) after having selected your graph:
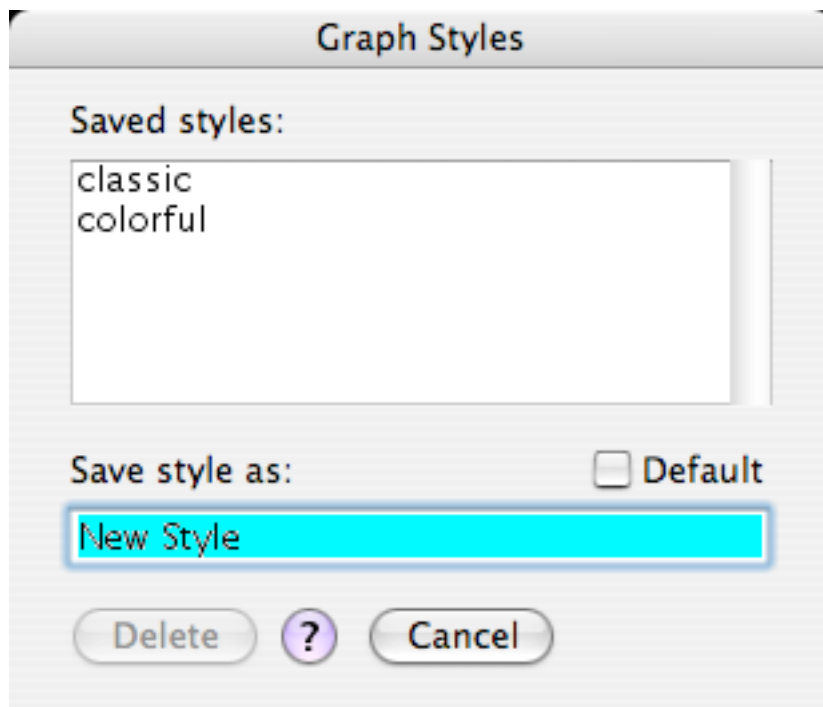
This box shows a list of the styles that are already saved in the current preferences file. You can delete one of these styles by selecting it and clicking **Delete**. To save a new style, enter its name and click **Save**. To load a style, select its name in the scrolling list and click **Load**. The name of the button changes from Save to Load when you move from the Style name edit field to the Saved styles scrolling list.

If you click the **Default** check box when saving a style, or if you define a style with the name "**Normal**", this style becomes pro Fit's default style. The next time you start up pro Fit, the first graph you create will use this style.

When you load a style, a dialog box appears, asking you to choose which parts of the style you want to apply to your graph:
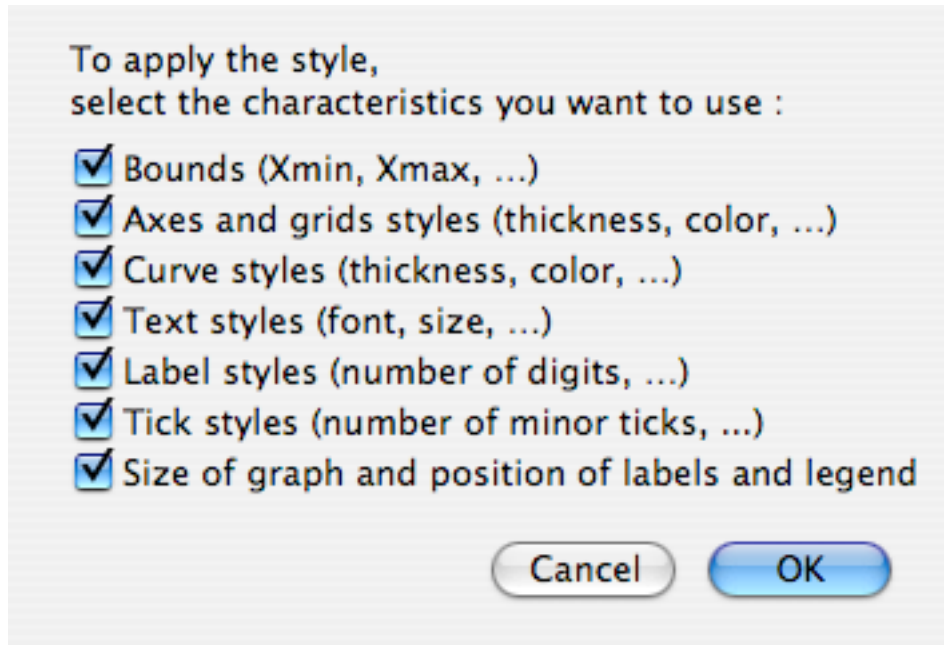
The characteristics of a style are:
- **Bounds**: The ranges of the graph, i.e. the minimum and maximum of all the axes; the positions of the first ticks; the distance between major ticks; the number of minor ticks.
- **Axes and grid styles**: The line thickness, dash and color of the axes, the frame and the grid; the distance of the labels from the axes; the location of the ticks (inside or outside).
- **Curve styles**: The line style of all plots, i.e. curves and data points.
- **Text styles**: The font, size, and text style of the labels.
- **Label styles**: The number format of the labels. The number of digits after the decimal point and the representation (exponential, auto, decimal) of the labels.
- **Tick styles**: The number of minor ticks, the axes scaling (logarithmic or linear) and whether the labels are visible.
- **Graph size**: The horizontal and vertical size of the graph (length of the coordinate axes) and the relative position of its labels and legend.

Graph coordinates and zooming

Normally you can look at coordinates and analyze data sets and function using the Preview window. However, options similar to the ones available in the preview window, although more limited, can be used when editing graphs.

Hold down the command and option key simultaneously and click and drag over a graph object. pro Fit displays the mouse location in the main axes coordinate system. The coordinates are displayed to the right of the cursor and in the bottom left corner of the drawing window.
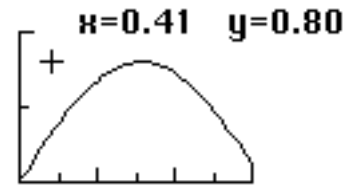
If you now press the shift key, you can select a part of the graph. The ranges of the graph will be changed to display only this part. This is useful for zooming in on some part of the plotted data set.
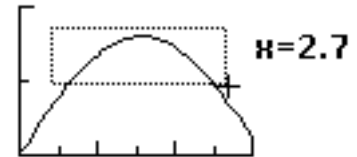
Shape properties

All shapes (objects) in a drawing window have "properties", such as their position or size. These properties can be read and set from a program, so it can manipulate shapes in a drawing window. For more information, see the documentation on GetShapeProperty and SetShapeProperties in pro Fit's on-line help.

Most of these properties can also be set and changed manually. For instance, when you move a shape to a new location, you change its position properties. Some of the properties can also be accessed by choosing "Shape Settings..." from the Draw menu. (You can also double-click most shapes for getting into the corresponding dialog box. When a window is in dialog mode, command-double-click the shape.)

The most important setting you can access through this box is the shape's name. This is a unique string attributed to each shape and used by programs for accessing the shape.

For more information, see Appendix A and Chapter 9 of this handbook.

Drawing windows in dialog mode

Drawing windows can be put into "dialog mode". In this mode, the window obtains the same background as dialog boxes. This is required when you want to create a complex dialog box using control shapes. You create and edit the control shapes while the drawing window is in its normal state. When you have finished, you switch the dialog window to dialog mode. In this mode, the drawing window cannot be edited anymore.
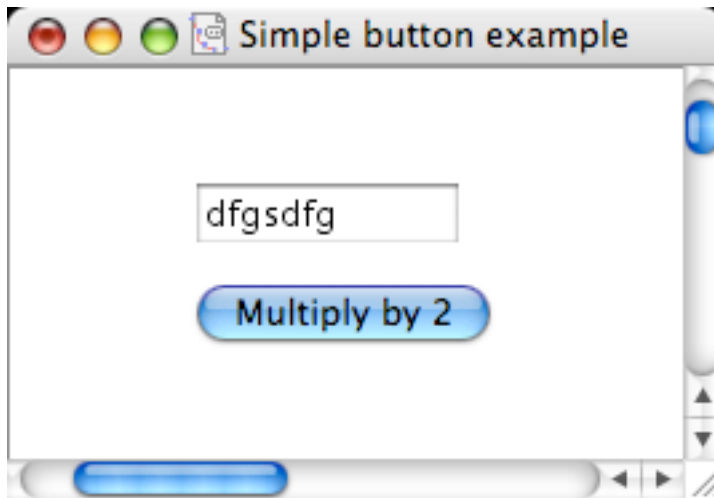
normal mode
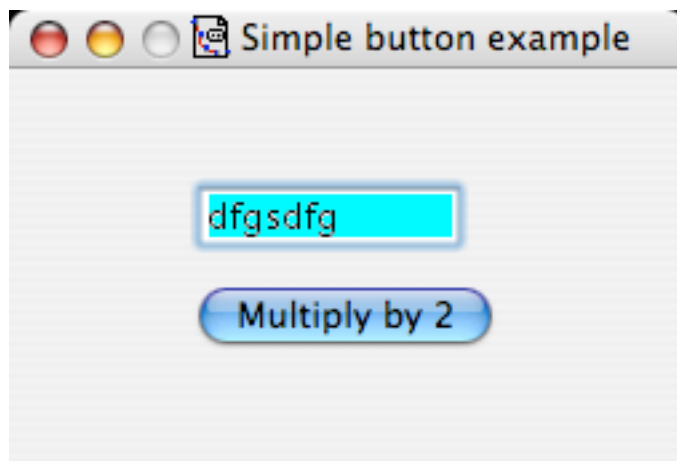
dialog mode

To switch a drawing window into dialog mode, hold down the control key while clicking anywhere into the window and choose "Display As Dialog". Alternatively, choose "Get Info..." from the File menu and check the option "Display As Dialog".

For more information on creating dialog boxes, see Chapter 9, section "Working with control shapes".

8   Fitting

This chapter describes what pro Fit does when you perform a *fit*.

'Fitting the parameters of a function to a data set' roughly means finding those parameters that make the function's curve follow the data points as closely as possible.

There are various possible definitions of the term 'as closely as possible'. The correct definition is often determined by the origin and characteristics of the data set to be fitted. For example, a data set might be subject to large errors in the x-coordinate and to smaller errors in the y-coordinates. The probability of incurring in a given measurement error can decrease in some known way when the magnitude of the error increases.

There are also various possible methods of looking for the best parameter set.

pro Fit provides a choice of different ways for "measuring the distance" of the data points from the function, as well as a choice of different methods to reach the best parameter sets.

The first part of this chapter deals with the definition and mathematical description of deviation functions and fitting algorithms, the second part shows you how to select these options in pro Fit and how to run a successful fit.

Mathematical background

In order to find the best parameter set describing a given measurement, it is necessary to establish a quantitative method to "measure the distance" between a data set and the function that should describe it.

This requires the introduction of weights for the data points and of probability distribution functions. They are described in the next sections.

Distribution functions and data weights

Consider a function $f(a_1,..,a_n,x) \equiv f(x)$ (we won't write explicitly the function parameters every time) and a measured data set $\{(x_1,y_1),..,(x_i,y_i),..,(x_N,y_N)\}$.
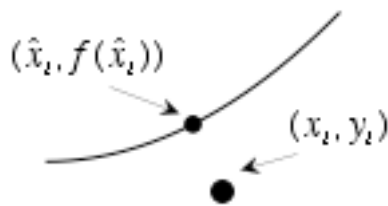
Let's assume that the function, with its "true" parameter set, correctly describes the quantity that was measured. We further assume that, when the data point (x_i,y_i) was determined, the "true" system (the one described by the function *f(x)*) was at the coordinates $(\hat{x}_i, f(\hat{x}_i))$. When the x-coordinate was determined, an inevitable experimental error occurred, and x_i was measured instead of $\hat{x}_i$. When the y-coordinate was determined, another inevitable experimental error occurred, and the measurement gave y_i instead of $f(\hat{x}_i)$.

In real life the true parameter set is not known. One has to measure it by measuring many data points at different coordinates and fit *f(x)* to the complete data set. This is the way we usually find a parameter

set which best describes the measurement. The parameter set obtained in this way is not the true (unknown) parameter set, but it should be a good approximation for it. (See the section on Error Analysis to find out how to estimate the errors of the fitted parameters.)

The fitted parameter set corresponds to a function $f(x)$ which maximizes the probability that the measured data set came from the system described by $f(x)$. To maximize this probability, we have to minimize the deviations between the measured data points and the function curve. This deviation can be defined in different ways, depending on the way in which the experimental errors are distributed, but it is usually a function of the weighted distances

$$d_{xi} = \frac{\hat{x}_i - x_i}{\sigma_{xi}} \qquad\qquad\qquad (1\ a)$$

$$d_{yi} = \frac{f(\hat{x}_i) - y_i}{\sigma_{yi}} \qquad\qquad\qquad (1\ b)$$

σ_{xi} and σ_{yi} give the magnitude of the *errors* expected when measuring the x_i and y_i, respectively. The role of these x- and y- errors is to define the correct scaling of the x- and y- deviations between a measured data point and the function that should describe it. The errors normalize the deviations, introducing dimension-less numbers d_{xi} and d_{yi}. Data points are weighted differently (given more or less importance) depending on their errors. A small error will magnify the importance of a given difference, a large error will make the normalized difference less important.

The distances d_{xi} and d_{yi} give the difference between measured coordinates and "true" coordinates. Obviously, we don't know the true coordinates, otherwise there wouldn't be any need for a fitting program in the first place. But we can estimate the true coordinates by minimizing some function of the distances d_{xi} and d_{yi}. This function describes the "difference" between the model function and the set of data points, and it is chosen in such a way that its minimization corresponds to the situation with the highest probability of producing the measured data set.

If the x- and y-errors are independent, a fitting algorithm must generally minimize a **mean deviation** χ_R of the type

$$\chi_R = \sum_i \left[R_x(d_{xi}) + R_y(d_{yi}) \right], \qquad\qquad\qquad (2)$$

where the functions $R_{x,y}$ are *deviation functions* that tell us in a quantitative way how bad it is that a certain (normalized) distance d is found for a data point. They are normally related to the **error probability distribution**. This is the function that gives the probability that a certain measurement error occurs. For example, $R_{x,y}$ can be the negative logarithm of the corresponding probability distribution for the distances d_{xi} and d_{yi}.

Minimization of χ_R as defined in Eq. (2) adjusts the function $f(x)$ in order to maximize the probability that the measured data set corresponds to an underlying "reality" described by the adjusted $f(x)$.

This is true as long as the following **assumption** is fulfilled: the measurement errors for each data point must be uncorrelated and described by probability distributions centered around the "true" values $(\hat{x}_i, f(\hat{x}_i))$.

The above assumption might appear harmless, but it is in fact more stringent than one would causally expect. For example, in most cases one tends to assume that the probability distribution is Gaussian, but the actual probability distribution for the measurement errors might be different, with a sizable

probability of finding larger errors from time to time, *i.e.* points that are clearly outside the expected trend ("outliers").

To allow for an analysis of such cases, pro Fit provides a set of deviation functions R which correspond to various error probability distributions.

The most common deviation function provided by pro Fit is the **squared deviation**

$$R(d) = d^2. \tag{3}$$

When using this deviation function, Eq. (2) becomes the **mean square deviation** between data points and function. Eq. (2) then corresponds to the negative logarithm of the probability of obtaining the data set in the presence of **normally distributed** measurement errors. The deviation function (3) corresponds to a **Gaussian** error distribution. In this case the probability density that a certain error occurs when measuring x_i or y_i is given by a Gaussian distribution (or normal distribution)

The next deviation function provided by pro Fit is

$$R(d) = |d|, \tag{4}$$

and corresponds to a two-sided **exponential** error distribution $\exp(-|d|)$. It leads to the calculation of a **mean absolute deviation** instead of a mean square deviation.

The deviation function

$$R(d) = \log(1 + \frac{1}{2}d^2), \tag{5}$$

corresponds to a **Lorentzian** error distribution $1/(1 + d^2/2)$.

The last deviation functions available in pro Fit are

$$R(d) = \begin{cases} c[1 - \cos(d/c)] & |d| < c\pi \\ c & |d| > c\pi \end{cases}, \tag{6}$$

with c=2.1 and

$$R(d) = \begin{cases} \dfrac{c}{6}\left\{1 - \left[1 - \left(\dfrac{d}{c}\right)^2\right]^3\right\} & |d| < c \\ \dfrac{c}{6} & |d| > c \end{cases}, \tag{7}$$

with c = 6. These deviation functions are called **Andrew's sine** (the derivative of (6) is $\sin(z/c)$) and **Tuckey's biweight**, respectively. They don't correspond to a particular probability distribution for the errors. They are designed to decrease the weighting of data points with very big errors (outliers) in order to allow a "robust" fitting through the more "reasonable" data points. It should be obvious that this procedure should only be used if you know your experiment and data set well enough, and we repeat the usual calls for caution!

Note that using the deviation functions (6) and (7) with another constant c is equivalent to changing all errors of the data points and the resulting mean deviation value by a constant factor.

Each term in the sum (2) describes a deviation between the measured data point (x_i, y_i) and the "nearest" point on the function curve $(\hat{x}_i, f(\hat{x}_i))$. The coordinate $\hat{x}_i$ must be chosen in such a way that each term in the sum (7) is minimized for each data point.
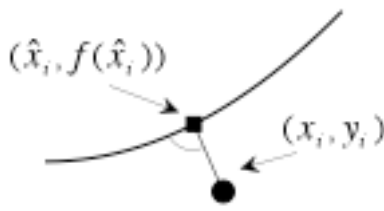
When the deviation function R is the squared deviation $R(d)=d^2$, then each term in (2) gives the square of the Euclidean distance between (x_i, y_i) and $(\hat{x}_i, f(\hat{x}_i))$. The term is minimized when the line connecting the data point to the function curve is perpendicular to the function curve. A fit-algorithm must thus adjust the function until the sum (2) of the squared perpendicular distances between data points and function curve reaches a minimum.

We refer to the literature for more detailed discussions of the above deviation functions. A short description is also found in the classical book by W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes - the Art of Scientific Computing*.

The mean square deviation: Chi-Squared

When squared deviation functions are used, (2) gives the mean square deviation, which is often called χ^2:

$$\chi^2 = \sum_i \frac{(\hat{x}_i - x_i)^2}{\sigma_{xi}^2} + \frac{(f(\hat{x}_i) - y_i)^2}{\sigma_{yi}^2} \tag{8}$$

The mean square deviation (chi-squared) is used when the measurement errors are described by a Gaussian probability distribution, and in this case the errors σ_{xi} and σ_{yi} correspond to the **standard deviations** of the Gaussian distributions.

The denomination "chi-squared" has become so common that it is often used to indicate the result of (2), and not only to indicate the particular case (8).

For the sake of simplicity, proFit follows this somewhat "dirty" convention and uses the denomination "chi-squared" when referring to the result of (2), even if deviation functions other than square deviations are used. The same is true for the predefined function `ChiSquared`, which can be used in proFit programs to retrieve the value of (2) obtained in the last fit.

Zero X-errors

In most experiments it is possible to determine the x-coordinate much more precisely than the y-coordinate. In such a case the x-errors can be assumed to be very small. The only way to minimize the mean deviation (2) is then to have $\hat{x}_i \equiv x_i$. The mean deviation function becomes much simpler:

$$\chi_R = \sum_i R\left(\frac{f(x_i) - y_i}{\sigma_{yi}}\right), \tag{9}$$

The function is evaluated at the x-coordinates of the measured points. The function value and measured y-coordinate directly give the normalized distance, when weighted with the measurement error.

The "usual case": Chi-squared and zero x-errors

In many experiments it is not only possible to make the x-errors so small that they can be considered zero. It is also common to have (or hope for) Gaussian distributed measurement errors. In this case we have to minimize a particularly simple expression for chi-squared:

$$\chi^2 = \sum_i \frac{(\mathrm{f}(x_i) - y_i)^2}{\sigma_{yi}^{\ 2}}, \tag{10}$$

Since this case is easy to handle from an algebraic and numerical point of view, many common fitting algorithms and applications work under the assumption that the mean deviation is the mean square deviation given by Eq. (10). A classical fitting algorithm that works on this basis is the Levenberg-Marquardt algorithm in its unmodified, original form (see below).

Error analysis and confidence intervals

Although some fitting algorithms (most notably the Levenberg-Marquardt fitting algorithm) do provide estimates for the error of the parameters, these estimates are often not sufficient or too imprecise.

pro Fit provides a general way for estimating the confidence intervals within which the "true" value of a fitted parameter can be assumed to lie with a certain probability level.

The influence of variations in the data points on the fitted parameters is analyzed with the help of a Monte Carlo simulation. For this purpose, synthetic data sets are generated starting from the points $(\hat{x}_i, f(\hat{x}_i))$ that were obtained in the fit (see above). For each of the original data points a simulated data point is generated by random variation around $(\hat{x}_i, f(\hat{x}_i))$ within the specified errors and using the specified error distributions. This produces a synthetic data set that effectively simulates a measurement. The simulation of the measurement is based on the function that was determined in the last fit (which is assumed to correspond to the underlying "reality") and on the measurement errors that were specified.

A short description of this error analysis technique is found in "W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes - the Art of Scientific Computing*".

For each of the synthetic data sets, a fit is performed. Once that all synthetic data sets have been fitted, the **confidence intervals** are calculated by analyzing the values obtained for each parameter. The confidence interval thereby corresponds to the range enclosing a given percentage of the values.

When error analysis is complete, the results are printed in the Results window and a list of the fitted parameters for each synthetic data set appears in a new data window. You can use the set of simulated parameters for further statistical analysis.

Fitting algorithms

In the previous section we gave a short overview of the most important mathematical tools used to establish criteria distinguishing a good fit from a bad one. Once these criteria are established, one can use them to analyze parameter sets, and to find out in which direction the best parameter set can be found.

The search for the best parameter set is the responsibility of a fitting algorithm, and pro Fit lets you choose between three different ones: The *Monte Carlo*, *Levenberg-Marquardt*, and *Robust* algorithms.

The algorithms differ by the method they use to orient themselves in parameter space and to find the location of the best parameter set.

The **Monte-Carlo** algorithm minimizes (2) with any definition of R by randomly varying the parameters and (if the x-errors are not zero) the set of x-coordinates $\hat{x}_i$ and looking for the smallest value of (2). This algorithm is often useful to scan parameter space and find good initial values for a Levenberg-Marquardt, or Robust fit.

The **Levenberg-Marquardt** algorithm minimizes the mean square deviations using (8). It finds at the same time the set of x-coordinates $\hat{x}_i$ and the function parameters that minimize the mean square deviations between the data points (x_i, y_i) and the function values $(\hat{x}_i, f(\hat{x}_i))$. When the x-errors are zero, the Levenberg-Marquardt algorithm minimizes (9).

The **Robust** fitting algorithm minimizes (2) with any definition of R by continually moving "downhill" in parameter-space until the bottom of a valley is found.

The **Linear Regression** and the **Polynomial** fitting algorithms are specialized for polynoms of 1st and nth degree. While the Linear Regression allows for x-errors (we use a straight forward algorithm if there are no x-errors), the Polynomial fitting algorithm is restricted to y-errors only.

The mathematics used by the various algorithms to perform their job is outlined in the next sections.

The Monte Carlo algorithm

This method randomly varies the parameters of a function within given intervals. When x-errors are defined, the algorithm also varies randomly the set of x-coordinates $\hat{x}_i$ while observing the given errors and error distributions.

For each random guess, χ_R is calculated according to Eq. (2) and the parameter sets corresponding to the smallest values of χ_R are remembered.

The strength of this method is also its biggest disadvantage. It looks for the best parameter set by shooting blindly inside the given region of parameter space. Although there is an option of letting this parameter space region follow the position of the currently best parameter set, this algorithm can only converge very slowly towards the best parameter set.

Its main use is to "scan" parameter space in order to find good parameter starting values for one of the deterministic fit algorithms, or to try to "jump out" of a local minimum where a deterministic fitting algorithm is stuck.

Since the algorithm is normally used for a first estimate of fitted parameters, it is not recommended to run it with non-zero x-errors – this merely slows down the algorithm without substantially increasing the accuracy of the estimates.

The Levenberg-Marquardt algorithm

The Levenberg-Marquardt algorithm is derived directly from the mean square deviation expressions (8) or (10) and cannot be used with deviation functions R other then the square deviation $R(d) = d^2$.

The Levenberg-Marquardt algorithm is in principle the fastest fitting algorithm available in proFit. Its performance, however, depends strongly on the behavior of the function to be fitted as well as on the selected starting parameters.

The classical version of the Levenberg-Marquardt algorithm does not allow for x-errors and minimizes the mean square deviation (10). The algorithm can be described in words as follows:

Starting from a given set of parameters, the mean square deviation χ^2 is calculated. Then the parameters are varied slightly to observe their influence on χ^2. From this, the direction in which χ^2 decreases most rapidly can be evaluated and a new set of parameters is chosen. This procedure is reiterated with this new set of parameters . When the minimum is near, the algorithm goes over to a more deterministic "guessing" at the position of the minimum and solves some equations to find it. The fitting stops when the value of χ^2 does not decrease anymore between successive steps.

The algorithm is described in W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes - the Art of Scientific Computing*, Second Edition, University Press, Cambridge, 1992.

When x-errors are specified, the algorithm is modified in such a way that it minimizes (8). It finds at the same time the set of x-coordinates $\hat{x}_i$ and the function parameters that minimize the mean square deviations between the data points (x_i, y_i) and the function values $(\hat{x}_i, f(\hat{x}_i))$.

The extensions to the Levenberg-Marquardt algorithm that allow the interpretation of x-errors are described in P.L. Jolivette, "least-squares fits when there are errors in X," Computer in Physics, Vol. 7, No. 2, 1993.

Partial derivatives

To fit a function of the type $y = f(a_1,..,a_n,x)$ the Levenberg-Marquardt algorithm needs the partial derivatives of the function with respect to its parameters. It uses the partial derivatives when it estimates the influence of the parameter set $\{a_i\}$ on χ^2. The partial derivatives f_i' are given by

$$f_i'(x) \equiv \frac{\partial f(a_1,..,a_n,x)}{\partial a_i} \tag{11}$$

and they are calculated for all x-coordinates $\hat{x}_i$ during every iteration.

> When you define your own function for fitting and you find that the fitting process is too slow, then you should define these derivatives explicitly (in the procedure called `derivatives`). If you do not define the derivatives yourself, pro Fit must calculate them numerically. This makes fitting considerably slower.

More information on how to define functions and their derivatives is given in Chapter 9, "Defining functions and programs".

Estimation of parameter errors

The Levenberg-Marquardt algorithm allows the determination of the standard deviations of the parameters. These are the values that are printed in the results window after a successful fit, under the heading "standard deviations". The standard deviation defines the region that contains 68.3% of the total integral of a Gaussian distribution.

Whenever fitting errors are specified, the standard deviation σ_{aj} of the parameter value a_j obtained after a successful fit is found from

$$\sigma_{aj} = [C_{jj}]^{1/2}, \tag{12}$$

where C_{ii} is the diagonal element of the **covariance matrix C**. The full covariance matrix of the parameters used in the fit is the inverse of a matrix **A**: **C**=**A**$^{-1}$.

When no fitting errors are specified, it is in principle not possible to calculate a covariance matrix. However, what pro Fit does in case of "unknown" errors is to calculate the covariance matrix using arbitrary error values of 1. It is then possible to make sense of this seemingly arbitrary covariance matrix by considering the chi-squared that was obtained in the fitting. For the standard deviations of the parameters, this amounts to calculating them from

$$\sigma_{aj} = [C_{jj} \; \chi^2]^{1/2}, \tag{12b}$$

If you need to work with the covariance matrix, and in general for any serious statistical analysis of a fit, you must define errors for your data points!

The matrix **A** is also called **curvature matrix**, and it is defined by the errors (standard deviations) of the data points and by the partial derivatives of the function with respect to the parameters. When x-errors are specified the derivative of the function with respect to x must also be calculated and the curvature matrix **A** is given by

$$A_{ij} = \sum_k \frac{1}{\sigma_{yk}^2 + \sigma_{xk}^2 \left(\frac{\partial f(x_k)}{\partial x} \right)^2} \left(\frac{\partial f(x_k)}{\partial a_i} \frac{\partial f(x_k)}{\partial a_j} \right). \tag{13}$$

If the x-errors can be considered to be zero, the curvature matrix **A** has the simpler form:

$$A_{ij} = \sum_k \frac{1}{\sigma_{yk}^2} \left(\frac{\partial f(x_k)}{\partial a_i} \frac{\partial f(x_k)}{\partial a_j} \right). \tag{14}$$

Loosely speaking, this matrix describes the propagation of the errors from the data points to the parameters. We refer to the specialized literature for more details.

If the x-errors can be regarded as zero, pro Fit lets you specify "unknown" y-errors. In this case, the y_i are assumed to be normally distributed, all with the same standard deviation σ. For fitting, σ_{yi} is taken to be 1 for all *i*. The "real" σ_{yi}^2 is then estimated from $\sigma^2 = \chi^2 / \nu$ (where ν is the number of degrees of freedom, i.e. the number of data points minus the number of parameters) and σ_{ai} is calculated from the expressions given above.

It is interesting to consider the case where a parameter reaches one of its limits during a fit. As you know, pro Fit lets you specify, for each function parameter, an interval of allowed parameter values. If a parameter is at one of the boundaries of this interval after a fit, its standard deviation cannot be calculated. The parameter is then considered to be constant (i.e. it is not a *free* parameter anymore). The standard deviations of the other parameters and χ^2 are calculated using the effective number of active parameters at the end of the fit. The results obtained are the same as those that would have been obtained by fitting with the parameter fixed at its limit from the start.

> The standard deviations of the parameters (and the covariance matrix) that are obtained in a Levenberg-Marquardt fit have a clear quantitative interpretation only if the errors of the data are normally distributed. If the data errors are not given, the calculations for evaluating the standard deviations of the parameters assume that the y_i are normally distributed and that the function is the correct description of reality.

Interpret the results carefully !

An alternative, more general way to estimate the errors of the fitted parameters is described in the section "Error analysis and confidence intervals".

The Robust minimization algorithm

This method minimizes χ_R (2) with any definition of R by continually moving "downhill" in parameter-space. Starting from some initial value, the parameters are varied and the resulting value of χ_R is calculated. From this, the algorithm finds the direction in which χ_R decreases and moves that way. Then it samples again the surroundings by varying the parameters. It stops when a minimum is reached.

When the x-errors are not zero, the $\hat{x}_i$ necessary for calculating the "minimal distance" between a data point and the function curve are calculated for each data point by an explicit minimization of the term $\left[R(d_{xi}) + R(d_{yi}) \right]$ in Eq. (2).

Minimization is performed with limited precision in order to save processing time. The $\hat{x}_i$ will be determined to an accuracy which is a fraction of the x-error specified for each point. pro Fit will also count the number of function calls it is using to determine one $\hat{x}_i$ and will stop after a maximum of 50 function calls (normally much less function calls (<10) are needed to find the minimum). This procedure introduces a small uncertainty in the determination of χ_R. However, the statistical significance of such an uncertainty will be limited, because the precision with which the $\hat{x}_i$ are determined is in any case much better then the errors of the data points.

> A robust fit with x-errors larger than zero will be considerably slower than the same fit performed with zero x-errors. When for zero x-errors evaluation of (9) requires a number of function calls equal to the number of data points, evaluation of (8) will require more or less ten times more function calls when x-errors are defined.

The Linear Regression algorithms

In this case we assume a straight-line model for the measured data with normally distributed errors.

$$y(x) = a + b\,x \tag{15}$$

A) If there are no x-errors and the y-errors are assumed to be known (σ_i is the uncertainty of y_i) equation (9) can easily be simplified. At its minimum the derivatives after the two parameters a and b vanish. This leads to a set of linear equations that are solved analytically:

$$a = \frac{S_{xx}S_y - S_x S_{xy}}{\Delta}, \quad b = \frac{S\,S_{xy} - S_x S_y}{\Delta} \tag{16}$$

using the following definitions:

$$S \equiv \sum_{i=1}^{N} \frac{1}{\sigma_i^2}, \quad S_x \equiv \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2}, \quad S_y \equiv \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2},$$

$$S_{xx} \equiv \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2}, \quad S_{xy} \equiv \sum_{i=1}^{N} \frac{x_i\,y_i}{\sigma_i^2}, \tag{17}$$

$$\Delta \equiv S\,S_{xx} - \left(S_x\right)^2$$

From these we are also able to calculate the variances of *a* and *b*, and the correlation coefficient between them:

$$\sigma_a^2 = S_{xx}/\Delta, \quad \sigma_b^2 = S\ /\Delta,$$

$$r_{ab} = \frac{-S_x}{\sqrt{S_x S_x}} \tag{18}$$

B) If the measurement shows errors in the x_i the minimization of (8) becomes more difficult, i.e. the set of equations derived for *a* and *b* are not linear any more. However, they are solved with numerical means, i.e. with a standard root finding algorithm.

Together with the fitting parameters and their variances the correlation coefficient r is calculated (Pearson's r). It takes a value between -1 and 1 depending on how much the x-values and the corresponding y-values are correlated. r = +1 if there is a complete correlation with a positive slope, r = -1 if there is a complete correlation with a negative slope, and r = 0 if there is no correlation at all.

The significance of the correlation is the "probability that |r| should be larger than its observed value in the null hypothesis" (x and y being uncorrelated). It ranges from 0 (= good correlation) to 100% (= bad correlation).

We refer to the specialized literature for more details.

The Polynomial fitting algorithm

Our model is the general linear combination of arbitrary functions

$$y(x) = \sum_{k=1}^{M} a_k F_k(x) \tag{19}$$

The functions F_k can be wildly nonlinear functions of *x*. "Linear" refers only to the model's dependence on its parameters a_k.

Once again we assume that the measurment errors σ_i of the *i*th data point are known. By defining the matrix **A** and the vectors **b** and **a** as

$$A_{ij} = \frac{F_j(x_i)}{\sigma_i}, \quad b_i = \frac{y_i}{\sigma_i}, \quad a_i \tag{20}$$

it is possible to describe the minimization equations in matrix form

$$(\mathbf{A}^T \cdot \mathbf{A}) \cdot \mathbf{a} = \mathbf{A}^T \cdot \mathbf{b} \tag{21}$$

The variances of the parameters can be found as the square root of the diagonal elements of the inverse matrix $\mathbf{A}^{-1}$.

To solve equation (21) we use the method of Singular Value Decomposition (SVD). It is a very robust algorithm for overdetermined as well as for underdetermined systems, although it is a little slower and needs more memory resources than solving the normal equations.

For further details see the literature listed below.

Goodness of fit

It is very important to know the quality of a fit; otherwise the minimizing parameters found are in general not meaningful. The goodness of fit, which is the probability Q that a value of chi-square χ should occur by chance, is calculated by the incomplete Gamma function

$$Q = gammq\left(\frac{N-M}{2}, \frac{\chi^2}{2}\right) \qquad (22)$$

It depends on the degree of freedom, defined as the difference between the number of measured points N and the number of varied parameters M.

If Q is large, e.g. > 0.1, the fit seems reasonable. If it is small, e.g. < 0.001, there might be something wrong.

Literature and suggested reading

W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes - the Art of Scientific Computing*, Second Edition, University Press, Cambridge, 1992.

P.L. Jolivette, "least-squares fits when there are errors in X," Computer in Physics, Vol. 7, No. 2, 1993.

The fitting process

General features

With pro Fit, fitting is a highly interactive process. You can decide which parameters have to be varied, set their starting values (estimates) and choose a fitting method. You can inspect the fitting process while it is running, and interrupt it if you don't like it. You can reiterate the process and change fitting algorithms..

 The fitting process starts from the parameter values given in the parameters window. You can change these values (click the numbers and edit them). The window also shows which parameters are to be fitted: Only those whose name is shown in **bold** face will be fitted (for these parameters the check box "Use for fitting", which appears when you select a parameter, is checked).

The following is the parameters window for the function Polynom. Among all the inputs, the one named x is also the default input value that is the independent variable for the function. All other inputs are parameters that influence how the function generates its return value from the default input value x. But for the parameter named 'deg', they will be all varied as fitting parameters.
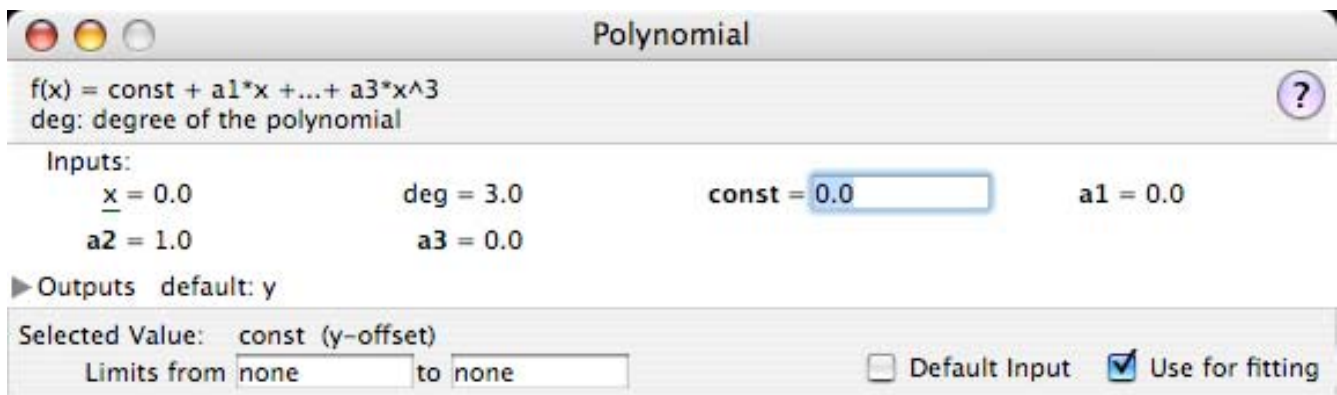
To change the fitting mode of a parameter (e.g. from 'fit' to 'not-fit'), click its name. It will switch from bold to normal or from normal to bold. Alternatively, you can click the check box **Use for fitting**, in the "selected parameter" field.

Some parameters can never be fitted. For example, it does not make sense to fit the degree of a polynomial. The name of such a permanently fixed parameter cannot be made bold by clicking it. The **Use for fitting** check box is disabled.

Parameters that can never be fitted are called *constant* parameters, those that are currently not fitted are called *inactive* parameters, and those that are currently fitted are called *active* parameters.**Parameter limits**

The value of a parameter can be limited to any specified interval by entering the boundaries of the allowed interval in the corresponding edit fields. The edit fields appear in the "active parameter" field once you select a parameter.   A parameter is not allowed to leave the specified interval during fitting, optimization of the function, or when you enter a new value in the parameters window.

See Chapter 5, "Working with functions and programs", and Chapter 8, "Defining functions and programs", for more information on how to set parameter limits in user-defined functions. If no limits are specified, the default values are -∞ and ∞ (-Inf..Inf).

During fitting, each parameter is constrained to the interval specified by the parameter limits.

Running a fit

Running a fit consists basically of three simple steps:

1. Choose the function to fit in the Func menu.

Add your own function to the Func menu if it is not already there.

2. Define which parameters you want to fit and their starting values.

You can do this in the parameters window as described above. Look at the function and the data set in the Preview Window to see how good your starting values are. Use the **Fitting-tool** in the Preview Window to "push" the function towards the data points.

The importance of good starting values depends on the function to be fitted. Some functions, like the Gauss function are more difficult to fit. A polynomial can be fitted with almost any starting parameter set.

Note: Some of pro Fit's built-in functions (such as Exp, Log and Power) provide algorithms for automatically choosing good starting values. For these functions, you can check the option "Guess Initial Parameters" in the fitting dialog box mentioned below and you don't have to set appropriate initial parameters before fitting.

3. Choose Fit from the Calc menu.

The Fitting window appears:

Fitting

Algorithm Levenberg–Marquardt   ☑ Auto search   Ranges...

Function   Polynomial   ☐ Guess initial parameters

Data

Data Window Table 1

☐ Selected data only

Input data

Data Column x–Value

Error type Zero

Output data

Data Column y–Value

Error type Unknown

▼ Plot

☐ Show last fit   1.0
☑ Show function   ☐ log
☑ Show data   ☑ auto

0.0

–1   ☑ auto ☐ log 1.0

☐ Error analysis:
☑ Print full description   ☐ Close window after fit
☐ Print active parameters only   (?) Done   Fit

Using this dialog box, you can set a number of fitting options. Once you are satisfied with them, click OK and fitting will start.

You can switch from one fitting algorithm to the other using the **Algorithm** popup menu. More details about particular options for each algorithm are given below.

The **Data Window** menu lets you select a data window (by default the foremost data window).

If **Selected rows only** is checked, only rows intersecting the current selection are used for fitting. Otherwise, all data in the X- and Y-columns will be used.

The **Data Column** menus define the data set coordinates x_i (input data) and y_i (output data).

The popup menus **Error type** let you specify the errors of your data. For the input data, choose **zero** to use zero x-errors (the usual case). In the output data, choose **unknown** if you don't want to specify y-errors – in this case, a value of "1.0" will be used as the error for all data points (regardless of the order of magnitude of the y-values) and all points will have the same weight in the calculation of χ_R (which is calculated with $\sigma_{yi} \equiv 1$). Choose **Constant** to set the standard deviation of all points to a given absolute value. Choose **Percent** if you want to enter the error as a fraction of the data value in %. If you have the errors stored in a column of your data window, then select **Individual** and choose the appropriate column in the pop-up menu that appears.

Make sure that the columns you select contain the correct error values in the correct positions. For each row in the table, there must be a one to one relationship between the values in the *x*-, *y*-columns and the values in the error columns.

The **Distribution** popup menu, which appears when you define errors, gives the error-probability distribution that will be assumed for the fit. This popup menu is dimmed if the Levenberg-Marquardt algorithm is used, because this algorithm only works with Gaussian error distributions.

The checkbox **Guess Initial Parameter** tells pro Fit to use a function specific algorithm for choosing appropriate starting values for the parameters. Such algorithms are available for some of pro Fit's built-in functions, such as 'Exp', 'Log', 'Power', as well as the 'Peaks' functions.

The resulting fitted function is shown in the section Plot. In addition, the numerical results are dumped to the results window. Check **Print active parameters only** if you only want to see the values of the parameters that were fitted. Use this option if your function has many parameters that you do not fit and you do not want all the values of inactive or constant parameters to clutter the results window. Check **Print full description** to get, for each fit, a header that describes the settings that were used for fitting.

Check **Error Analysis** if you want to obtain more information on the accuracy of the fitted parameters. Confidence intervals for each fitted parameter will be determined by a Monte Carlo method that simulates a large number of fits with a series of synthetic data sets. More about this in the Error Analysis section, below.

To start fitting, click **Fit**. Fitting can run for fractions of a second or for hours, depending on the execution speed of the function you selected, the number of parameters to fit, and the number of data points. The results of the fit appear in the result window. You might want to choose its name from the windows menu and position it in a comfortable place before running a fit. You can let pro Fit always bring the results window to the front after a fit by using the Preferences... command.

To speed up fitting when you are using one of your functions, you should define the function's partial derivatives with respect to its parameters. The section "The role of the partial derivatives" below gives more information on this topic.

You can interrupt fitting by holding down the command key ( ) and the period-key (.) simultaneously.

Note that pro Fit can run any fit in the background, this means that you can work with another application while pro Fit is fitting. You may want to place pro Fit's progress window in a corner of your screen to watch what is going on.

Inspecting the progress of a fit

During lengthy fits, you can inspect what is going on and see if the fitting algorithm is behaving correctly. pro Fit displays information on the current fit in its progress window:
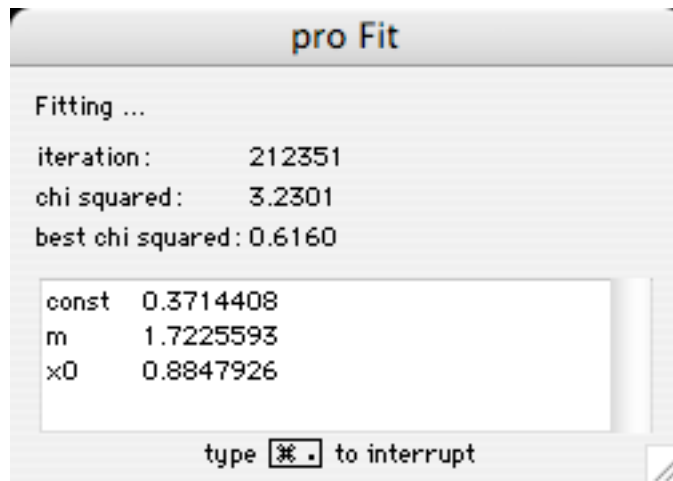
This window lists the total number of iterations, the current values of chi squared, and the current values of the best parameter set.

You can see the progress of the fit graphically if you open the Preview Window and check the Show Function check box. During a fit, pro Fit will periodically draw the function corresponding to the best parameter set. This allows you to see how the function approaches the data set during a successful fit. Because of this previewing feature, you will notice soon enough if the fit is not converging correctly, and will then be able to interrupt it.

> If your function performs many lengthy calculations, redrawing the function periodically can slow down the fit. Hide the Preview Window, or uncheck "Show Function" if fitting speed matters.

Error analysis and confidence intervals

Check **Error Analysis** in the Fit dialog box to get more information on the confidence intervals of the parameters.

When Error analysis is checked, two more edit fields appear in the Fitting Setup dialog box.

The Error Analysis algorithm simulates a number of data sets equal to the value specified in the **iterations** edit field. For each iteration, the corresponding parameter set will be determined by the fitting algorithm you selected (either the Robust algorithm, or the Levenberg-Marquardt algorithm).

> You should always use the Levenberg-Marquardt algorithm when performing error analysis. Using the Robust algorithm is not recommended because this algorithm is inherently slower than the Levenberg-Marquardt algorithm. Since error analysis can need thousands of iterations, the convergence speed of the algorithm is very important.

All parameter sets generated during error analysis will be collected and displayed in a data window once Error Analysis is completed. You can then use them for a more complete analysis of the distribution of each parameter.

Based on the simulated parameter sets, pro Fit estimates confidence intervals. You must specify which confidence interval you want pro Fit to calculate by entering the corresponding probability in the **confidence intervals** field. pro Fit calculates a confidence interval in such a way that the given percent of the simulated parameter values are contained inside it.

During error analysis, pro Fit shows the status of the calculation in its progress window:
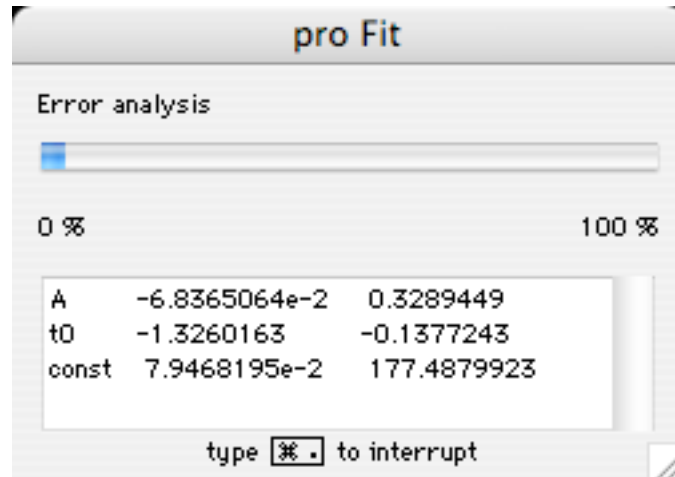
The window shows the status of the calculation and the confidence interval estimations based on the currently available data. The calculation is a Monte Carlo calculation, so the boundaries of each confidence interval will converge slowly and randomly towards some stable values.

If you want to see what happens during error analysis and your function draws itself fast enough, open the Preview Window and make sure "Show function" is checked. pro Fit will redraw the function periodically during error analysis and you will be able to see how the fitted function changes depending on the simulated data sets which are generated randomly. However, doing so will waste time for drawing the function and slow down the error analysis procedure. Hide the Preview Window, or uncheck "Show function" to make the error analysis procedure as fast as possible.

Fitting results

When fitting is completed, a summary of the results of the fit is displayed in the results window. Depending on which fitting algorithm you used, the data printed to the results window can vary slightly.

You may often want to transfer the values of the fitted parameters to the parameters window to use them as starting values for a further fit. Choosing **Get Fitted Params** from the Calc menu transfers the best set of parameters to the parameters window.

The results of a fit are made available to custom functions and programs through a set of predefined functions used for accessing the fitted parameters, the confidence intervals, the value of chi squared, and, for the Levenberg-Marquardt algorithm, the full covariance matrix. See pro Fit's on-line help for more details.

If you want to save the parameter sets obtained in every single fit, store them in a dedicated data window. You can copy them from the parameters window and paste them into a single row of the data window, or you can write a small macro (a pro Fit program) that transfers the fitted parameters directly to their data window. See chapter 9 "Defining functions and programs" to see how to do this. An

example program for transferring parameter values to a data window is found on the pro Fit distribution disks.

Using the various fitting algorithms

pro Fit provides three different fitting algorithms: The *Monte Carlo*, *Robust*, and *Levenberg-Marquardt* algorithms. They are described in the preceding section.

The following sections describe how each of these fitting algorithms is used, and what particular options you can set for each algorithm.

Using the Levenberg-Marquardt algorithm

To start a fit with the Levenberg-Marquardt algorithm, choose **Fit** from the Calc menu after having selected the appropriate function from the Func menu. The Fitting Setup dialog box appears with Levenberg-Marquardt pre-selected in the Algorithm popup menu. :

See the preceding section for a description of this dialog box.

When you define errors, the **Distribution** popup menu is dimmed and set to a Gaussian distribution. The Levenberg-Marquardt algorithm can only work if the errors of the data set are normally distributed.

The Levenberg-Marquardt fit stops running when the chi-squared determined from the current parameter set doesn't decrease appreciably anymore from one iteration to the next.

When finished, the parameter values and their standard deviations are printed to the results window. If you need to access the complete covariance matrix, you can define a program that uses the predefined function `CovarMatrix`. See pro Fit's on-line help for more details on how to use this function.

Using the Robust minimization algorithm

To run a Robust fit, choose "Robust" in the algorithm popup menu of the fit dialog box. This dialog box appears when selecting "Fit" from the Calc menu, and it was described above.

Using the **Distribution** popup menu, which appears when you define errors, you can select the error distribution that best describes your experiment. Robust fitting will deserve its name if you select a distribution that diminishes the importance of outliers (like Andrew's sine or Tuckey's biweight).

When finished, the resulting parameter values are printed to the results window. This algorithm does not determine a "standard deviation" for each parameter, like the Levenberg-Marquardt algorithm does. To obtain error estimations you have to run a Levenberg-Marquardt fit after the Robust fit converged, or you have to check the **Error Analysis** check box and perform a Monte Carlo analysis. See the corresponding section for more details.

Using the Monte Carlo algorithm

To run a Monte Carlo fit, choose **Monte Carlo Fit** from the Calc menu or chose Monte Carlo in the Algorithm popup menu of the Fitting Setup dialog box.

When you do this, two more items appear to the right of the Algorithm popup menu (Please refer to the beginning of this section for a basic discussion of the Fitting window.)

Clicking **Ranges...** presents another dialog box where you can define the ranges within which the parameters can be varied:
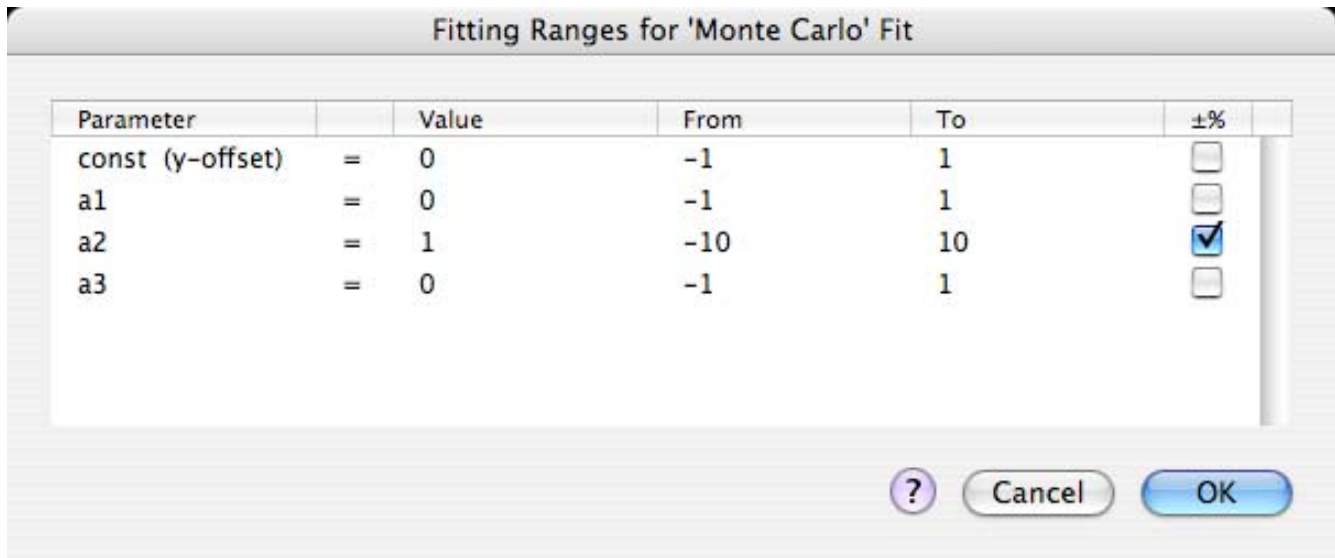
By default, these ranges are the ten percent deviations of the starting value of the parameter (or –1 and 1 if the starting value is 0).

Checking **Auto search** tells pro Fit to make a more flexible search for the best set of parameters.

The Auto search check box determines whether the limits within which the parameters are varied will be kept fixed (auto-search unchecked) or if they will be adapted during the fit (auto-search checked). In the latter case, the limits will be shifted after every iteration to keep them around the best parameter set. (Note that the parameters are never allowed to leave the parameter limits defined in the parameters window.)

The Monte Carlo Fit runs until you interrupt it by pressing -'.'. If you don't stop the fit yourself, the Monte Carlo Fit runs for ever.

The three best sets of parameters are displayed in the results window after you interrupt the fit.

> The Monte Carlo fit slows down exponentially when the number of parameters to be fitted is increased.

Using the Linear Regression algorithm

To run a Linear Regression fit, choose "Linear Regression" in the algorithm popup menu of the fit dialog box. This dialog box appears when selecting "Fit..." from the Calc menu, and it was described above.

As the name indicates, this algorithm forces you to select the Polynomial function of degree 1, with both parameters being fitted. It assumes a Gaussian distribution of errors. X-errors and Y-errors are possible.

When finished, the parameter values and their standard deviations are printed to the results window. Additionally, the correlation coefficient r is calculated, as well as its significance, which is the probability that |r| should be larger than its observed value in the null hypothesis (x and y being uncorrelated).

Using the Polynomial fitting algorithm

To run a Polynomial fit, choose "Polynomial" in the algorithm popup menu of the fit dialog box. This dialog box appears when selecting "Fit..." from the Calc menu, and it was described above.

As the name indicates, this algorithm forces you to select the Polynomial function of any degree. It assumes a Gaussian distribution of errors. Only Y-errors are possible.

When finished, the parameter values and their standard deviations are printed to the results window.

Fitting multiple functions and x-values

You may sometimes want to fit *simultaneously* several functions ($f_1 .. f_q$) with one or more common parameters. Or in other words, you may have a function with many outputs and use it to fit a multi-dimensional set of data. Or you may want to fit a function that does not depend on a single *x*-value but on a set ($x_1, x_2 ... x_p$) of *x*-values. Or you might even encounter a combination of these two cases.

In the most general case, you have q functions (or q different outputs $f_1, f_2, … f_q$) , each of them depending on one or more inputs. Among the inputs, some are used as independent variables, corresponding to the quantities that were varied during a measurement, others as parameters. Each output is calculated using one or more of the various inputs. You can see this as a set of functions that can share one or more parameters and that depend on any subset of the indepenent variables:

$$y1 = f1(x1, x2 ... x_{p_1})$$
$$y2 = f2(x1, x2 ... x_{p_2})$$
$$…$$
$$yq = fq(x1, x2 ... x_{p_q})$$

For each function or output value, you have a set of data points that should be described by it. Now you want to fit all these functions simultaneously.

pro Fit provides a dedicated tool for this type of multidimensional fitting.

There are several methods of tackling this kind of problem with pro Fit. This handbook has not yet been updated to cover them, but pro Fit's online help provides extensive documentation of this feature. To view it, chose pro Fit Help from the Help menu and enter the search string "multiple fitting".

General hints for fitting

Starting parameters

As already pointed out, the success of a fit often depends critically on the choice of a good set of starting parameters. Bad starting parameters can cause convergence to a false (i.e. local) minimum of

the mean deviation χ_R. It is good practice to always try to figure out reasonable values for starting parameters. Some of the functions are able to take their own guess at a good set of starting parameters, but in many cases a human is still the best judge.

Redundancy of parameters

Sometimes a fit converges slowly or is even stopped with the cryptic error message '**A singularity occurred**'. This can be caused by badly chosen starting values for fitting. However, this error is often a consequence of poorly defined or redundant parameters. For example, consider the exponential function

$$y = A \times \exp\left(\frac{-(x-x_0)}{t_0}\right) + const.$$

This function has four parameters: A, x_0, t_0 and $const$. However, the parameters A, t_0 and x_0 are not independent, as it is easily seen when writing (5) in factors:

$$y = A \times \exp\left(\frac{x_0}{t_0}\right) \times \exp\left(\frac{-x}{t_0}\right) + const .$$

The first two factors (A and $\exp(x_0/t_0)$) both have the same influence on y. A change of x_0 can be compensated by a change of A. These parameters are redundant. When trying to fit them simultaneously, the fit fails.

> Another problem often encountered during fitting is caused by the 'poor' definition of a parameter. Example: If you are trying to fit the data points ($x_1 = 1$, $y_1 = 2.01$), (2, 3.99), (3, 6.00), (4, 8.02), (5, 9.98), (6, 12.00) to a polynomial of second or higher degree
>
> $$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... ,$$
>
> you will get a very poor estimation of the parameters a_2, a_3, ... because your data points are nearly on a straight line and are sufficiently described by the parameters a_0 and a_1. The standard deviations of the coefficients a_2, a_3, ... will be accordingly large.

The errors of the data set

When using errors (standard deviations) for your data, it is useful to keep some points in mind:

- Multiplying all errors of your data points with a common factor does not affect the results of fitting, but changes the estimate of the standard deviations or the confidence intervals of the fitted parameters.
- Changing the relative errors of your data points affects the numerical weight of the data points. Example: If you have a large number of points in one area (e.g. between $x = 1$ and 2) and just one or two points far out (e.g. at $x = 50$), it is necessary to decrease the error for these 'lonely' points if you want to force the function to come close to them.
- When plotting a curve in a graph with a logarithmic y-axis, a deviation of the curve from a small y-value appears much larger than the same deviation from a larger y-value. If this astonishes you, it is probably because your measurement errors are proportional to the measured value. When plotting a fit on a graph with a logarithmic y-axis, the errors of the y_i are often given in percent. This results

in smaller deviations from points with small y-values. Here is an example of logarithmically plotted data with fits using percentage errors and constant errors.
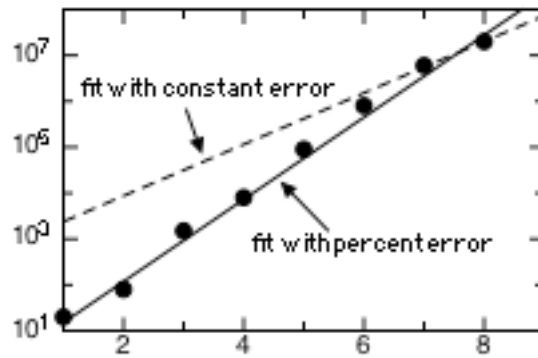
A fit with percent errors gives a more satisfying visual agreement between curve and data. Obviously, for serious data fitting you should always specify the real measurement error you expect for every data point.

9 Defining functions and programs

pro Fit allows you to define *functions* and *programs*:
- A **function** is added to the menu 'Func'. It behaves like any of pro Fit's built-in functions and you can use it for fitting, plotting, etc., see Chapter 5, "Working with functions".
- A **program** is added to the menu 'Prog'. A program performs a sequence of tasks. Programs can be used for scripting pro Fit.

Both, functions and programs, can be defined in the same syntax, which is based on the Pascal programming language. In addition to this, programs can be written in Apple Script.

All commands that can be given to pro Fit using its menus, can also be issued through pro Fit's program definition language or through AppleScript. You do not need to know much about the syntax of these "programming languages" in order to do this. The command that corresponds to any user-action can be generated automatically by switching on "recording", either in pro Fit, or in Apple's Script Editor, or other equivalent scripting utilities.

Programs can be considered to be "macros" that can be used to automate tasks. However, a pro Fit program can do much more than what you would normally expect a macro to do, such as complicated calculations and data transformations.

Here is a small list of what functions or programs can do:
- Calculate any kind of numerical value, even if it cannot be expressed in a closed mathematical formula.
- Access the data in a data window, write results into the results window, use dialog boxes and alert boxes.
- Execute any command from pro Fit's menus, open and save files, create and close windows.
- Run fitting operations and predefined numerical algorithms and retrieve their results.
- Create graphs and other drawings in a drawing window using a precise, floating point coordinate system.
- Access the properties of drawing objects in drawing windows, and mange buttons, check boxes, popup menus, or other interface elements that can be drawn there.

Note: All the above can also be done from a plug-in – a piece of code generated by your favorite compiler. If you are used to programming your own code for data or function analysis, you can consider pro Fit as a big library offering routines for numeric analysis, data input/output and high resolution graphics. Information on how to create a plug-in is found in Chapter 10, "Working with External Modules".

When you are defining your functions and programs within pro Fit, they are translated ('compiled') into native computer code when they are added to pro Fit's menus. This code can be executed very quickly by your Macintosh.

Simple programs and functions can be defined very easily and quickly.

Even very complicated programs can be defined without much work by simply recording your activities using pro Fit's automatic macro recording feature.

This chapter first gives a short overview on the principles of programming in pro Fit. It then explaines the automatic macro recording feature, and finally it lists the features of pro Fit's built-in compiler in detail. At the end, it explains how to save programs and functions as plug-ins for later use.

Simple examples

Defining functions

Imagine you want to analyze a function of the form

$$y = B \sin(x) \ln(x) + D \qquad\qquad (8.1)$$

with the parameters B and D. To define it in pro Fit:

1. Choose 'New Function' from the File menu.

This opens a new, empty function window.
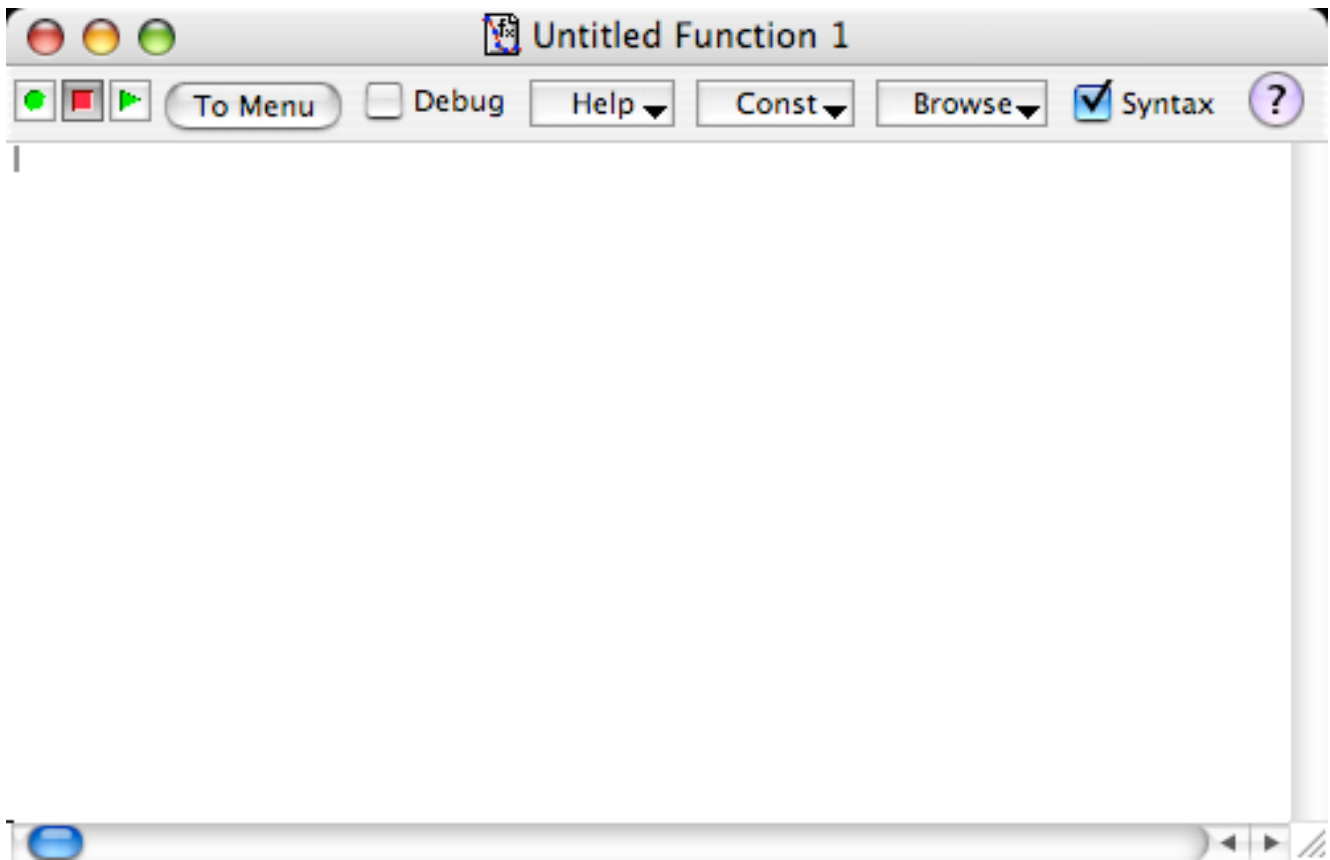
2. Enter the formula of your function in the new window.

In the simplest example, you could simply enter the formula as given above, using a[1], a[2] for the parameters a

```
a[1]*sin(x)*ln(x) + a[2]
```

3. Click 'To Menu in the function window or choose 'Compile & Add To Menu' from the Customize menu.

pro Fit analyses the contents of the window. Since you have entered a simple mathematical expression using the name *x*, pro Fit assumes that you want to define a function. Your formula is translated into a Pascal-like function definition, it is compiled and added to the menu 'Func'.

The function window now shows a minimalists Pascal definition of your function:
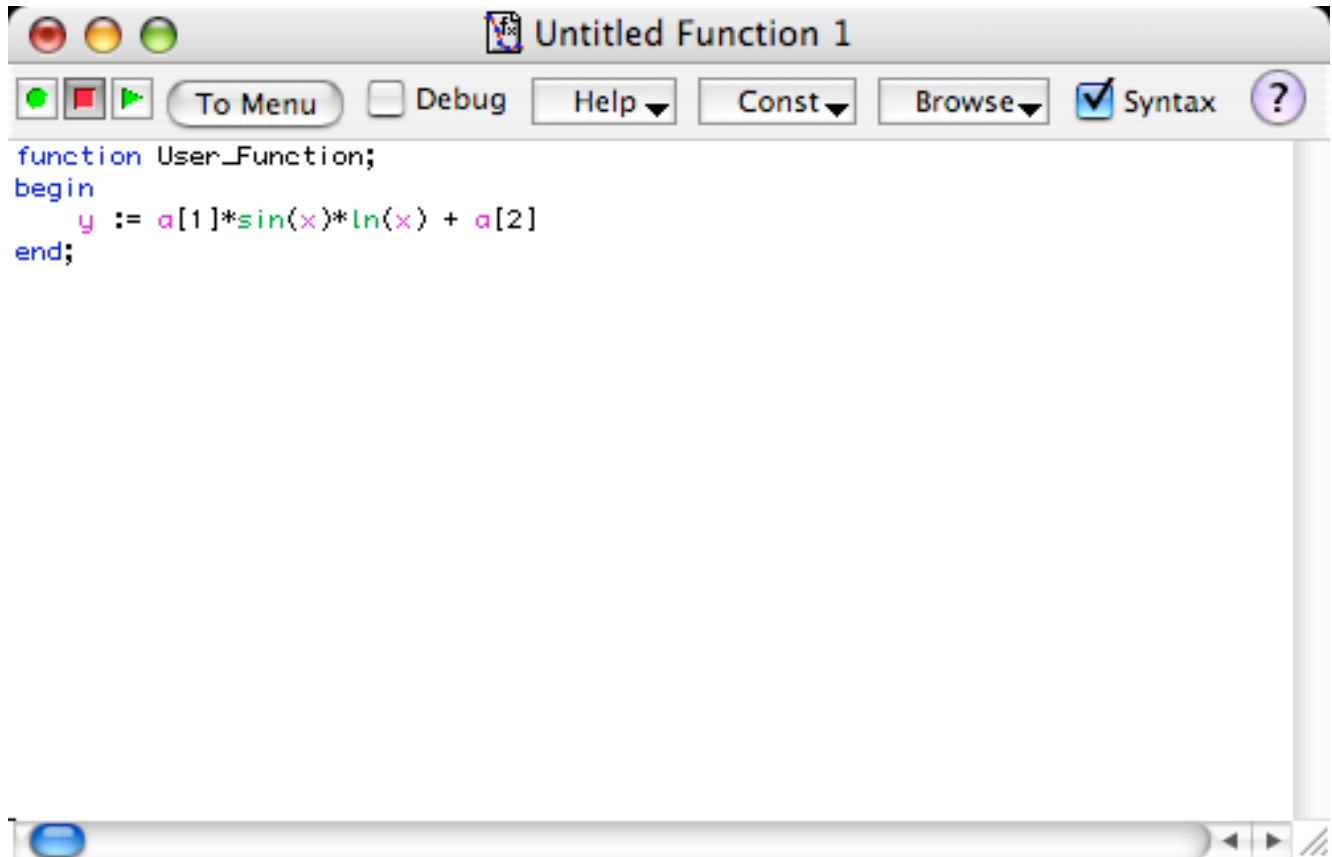
The new function appears under the name "User_Function" in the menu 'Func'. It is automatically selected and the parameters window shows its parameters a[1], a[2].

After adding the function to pro Fit, you can change its parameters in the parameters window. You can plot the function, use it for fitting, calculate a table of its values, etc. (To view the function in the preview window, make sure that the option "Show function" is checked.)

The above method is an abbreviated way for entering functions: you simply enter the function's expression and pro Fit translates it into a Pascal function definition before compiling it. In many situations you will, however, want to write or edit the function definition directly. Therefore, let's have a closer look at the minimalist function definition appearing above:

```
function User_Function;
begin
    y := a[1]*sin(x)*ln(x) + a[2];
end;
```

The first word of our example is `function`. It tells pro Fit that the definition of a function follows. The next word (`User_Function`) gives the name under which the function will appear in the Func menu.

The function's actual definition is given between the keywords `begin` and `end`. The function's value is calculated and then assigned (by the `:=` operator) to the function output named `y`. The variable `x` contains the function's default input value and `a[1]`, `a[2]` etc. are two additional input values that normally play the roles of function parameters.

`a` is a predefined array that represent the function parameters (or any additional input value past the standard x-value). The parameters can be accessed by their index, i.e. `a[1]`, `a[2]` etc. Instead of using `a[i]` for the parameters, you can also use parameter names of your own by declaring them (as in standard Pascal) in the header of the function

A less minimalist definition for our function could be

```
function LogSine(B,D:real);
begin
    if x <= 0
       then y := D
       else y := B*sin(x)*ln(x) + D;
end;
```

which uses two names for the two additional inputs (*B* and *D*) and provides a mechanism to take into account the possibility that the x-value can be netagive. Our sample function above was not defined for x<=0. It would generate a run-time error if used in calculations with negative *x*-values. But since the function converges to y=*D* for x=0 we can just decide that the output value is *D* for all negative values of *x*.

Note that you can insert additional spaces or lines anywhere between keywords.

The new version of the function (which now has the name 'LogSine') shows how you can define names for the parameters *B* and *D* (simply list them between brackets after the function name as shown) and use the `if` statement for conditional execution. The *if-statement* takes the general form

> `if` *condition* `then` *do this* `else` *do that*

'*do this*' is executed if the condition is met, '*do that*' if it is not met.

If you work with your function more often, you might want to make sure that the Parameter window shows reasonable default values for the parameters and a short description of what the function does. Here is a final and more complex definition implementing this (note that texts between curly brackets ('{' and '}') are used as comments and are ignored):

```
function Myfunction(B, D: real);
description
    { text to appear in parameters window }
    'x > 0:   y = A sin(x) ln(x) + B',
    'x <= 0:  y = B';
inputs
{ names and defaults for the parameters }
    B := 1,active,'B (the amplitude of this function)';
    D := 0,inactive,'D (a constant offset)';


begin
    if x <= 0
        then y := B
        else y := B*sin(x)*ln(x) + D;
end;
```

When you add this function to pro Fit, its parameters window looks like this:
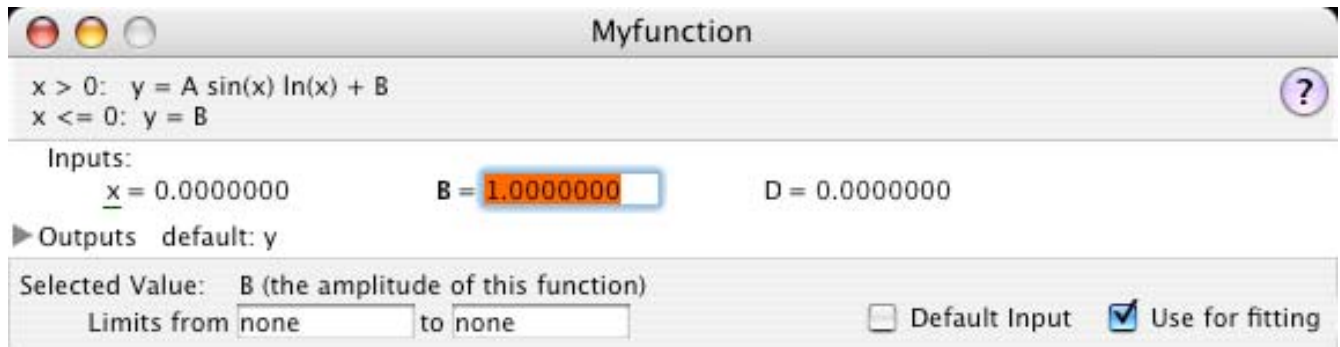
In the last version of our sample function two additional elements have been added:
- A keyword `description` followed by two texts between quotes ('...'), which appear at the bottom left of the parameters window.
- A keyword `inputs` (in previous pro Fit versions, `defaults` was used), which is followed by additional information for each input that appears in addition to x, and takes the form: `a[i] := value, mode, name, lowLimit, highLimit`, where `value` is the default value of a parameter, `mode` its default fitting mode (it can be 'active' (i.e. the parameter will be fitted), 'inactive' (will not be fitted) or 'constant' (cannot be fitted)) and `name` (its name between quotes (' ') used in the parameters window). Instead of using `a[i]` you can also directly use the name that you declared between brackets after the function name. Using the keyword `inputs`, you can also define a range of acceptable values for an input, given in the optional parameters `lowLimit` and `highLimit`. See the detailed description of the `inputs` keyword, later in this chapter.

Once you have successfully defined a function and you have added it to the Func menu, you can save it as a plug-in. A plug-in is a file that contains the computer code for your function and that can be loaded by pro Fit at start-up, or at any other time. Go to the last section of this chapter for more information on plug-ins.

Defining programs

Programs are generally used to create or transform data in a data window or for scripting pro Fit operations. In the following we give some very simple examples of programs.

In a first step, we will write a program that fills the first column of a data window with the powers of two: 2, 4, 8, 16, etc.

1. Choose New Function from the File menu.

This opens a new, empty function window.

**2. Enter the definition of your program in the new window.**

Enter the following definition:

```
program PowersOf2;
var i: integer;
begin
  NewDataWindow;
  for i := 1 to nrRows do
     data[i,1] := 2 ^ i;
end;
```

Note that a program definition starts with the keyword `program` followed by its name. After that we first have a variable declaration for the variable `i`, which is of type integer.

The body of our program between `begin` and `end` starts with the call `NewDataWindow`, which tells pro Fit to open a new, empty data window. Then follows a so-called *for-loop*, which takes the general form

    `for` *variable* `:=` *startValue* `to` *endValue* `do` *statement*`;`

A for-loop executes its statement for all integer values of its variable between startValue and endValue. If startValue equals endValue, the for-loop is executed only once. If startValue is larger than endValue, the for-loop is never executed.

The end value in our for-loop is `nrRows`, `nrRows` is always equal to the number of rows in the current data window.

The statement in our for-loop is an assignment (`:=`) to the array element `data[i,1]` that corresponds to the ith data cell in the first column of the current data window. The expression `2^i` stands for 2^i (you can also use `2**i` instead of `2^i`)

3. Click 'Add' in the function window or choose 'Compile & Add To Menu' from the Prog menu.

The program is transformed into computer executable code (it is compiled) and its name appears in the at the end of the menu 'Prog.

4. Choose PowersOf2 from the menu 'Prog'.

The program is executed. It opens a new data window and fills its first column with the desired values.

Our next example program is somewhat more complex. Imagine you have a data window with some data in the first column. You want to write a program that fills the second column with the square root of the values of the first column. You want to take some special cases into account:
  • If a cell in the first column is negative, the corresponding cell in the second column should be 0.
  • If a cell in the first column is empty, the corresponding cell in the second column should be empty too.
  • If any cell in the first column was empty, the program should give the user a warning when it has finished.

The program which does this task looks like this:

```
program MakeRoot;
var i: integer;              {the row counter}
    doAlert: boolean;   {true if a cell}
                        {was empty}
begin
  doAlert := false;
  for i:=1 to nrRows do
    if DataOK(i,1) then {if cell not empty}
      if data[i,1]>=0
        then data[i,2] := sqrt(data[i,1])
        else begin
              data[i,2] := 0;
              doAlert := true;
            end;
  if doAlert then
    Alert('Some data was negative');
end;
```

This program shows some additional features of the definition syntax:
  • An additional variable of type `boolean` has been introduced. A `boolean` variable can take the values `true` or `false` which can be used in `if` statements.
  • Before accessing the data in a cell, we test if there is really a number in this cell. This is done with the function `DataOK(r,c)`, which returns true if the cell in row `r` and column `c` contains a valid number. If the cell is empty or if it contains text, it `DataOK` returns false.
  • The innermost `if` statement (`if data[i,2]>=0`) has two statements in its `else` branch. They are grouped by the keywords `begin` and `end` to make it clear that they both belong to the `else` statement.
  • At the end of the program an `if` condition checks whether any data was negative. If there were negative numbers in the input column, the procedure `Alert` is called. `Alert` takes one argument, a string (i.e. a text between quotes). It displays an alert box that shows this string. Here is the alert box that appears in case negative numbers are found when the above program is executed:
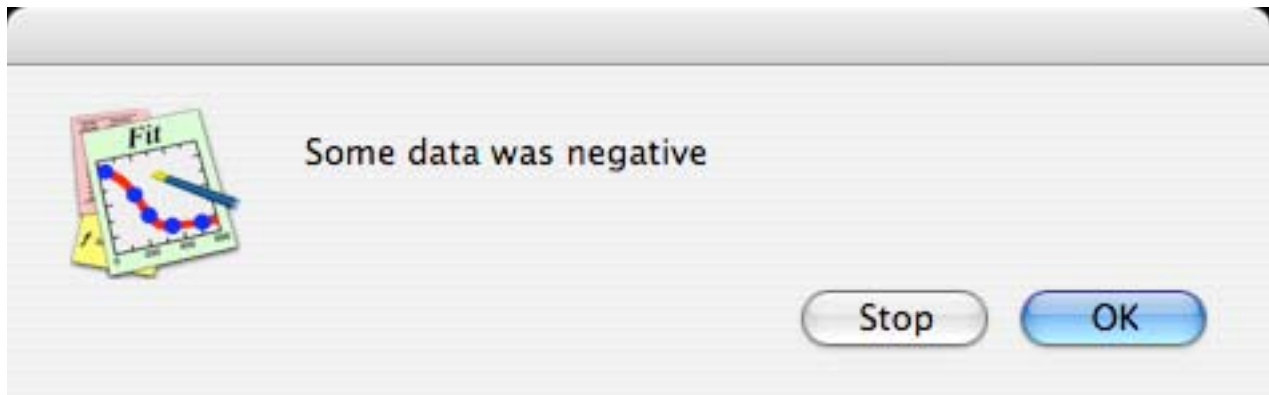
This alert box has two buttons: 'Stop' and 'OK'. If you click **Stop**, the execution of your program is immediately aborted. If you press **OK**, the execution of your program continues. For our sample program, it will not make any difference if you press Stop or OK: when the program calls `Alert`, it is at its end anyway.

You can now add the sample program `MakeRoot` to the Prog menu (click **Add** in the function window). Then prepare a data window with some data in its first column and run MakeRoot (by choosing MakeRoot from the Prog menu).

A shortcut

As mentioned at the beginning of this chapter, you can abbreviate the definition of a function by simply entering its expression (using x and the parameters a[1], a[2], etc.) in a function window. When you click the button "To Menu" or choose "Compile & Add to Menu" from the 'Customize' menu, pro Fit scans the contents of the text window.Compile & Add to Menu If it encounters a simple expression using x, it assumes that you want to define a function and adds the corresponding Pascal syntax around your expression.

You can use the same mechanism for defining programs. For example, you can simply enter the following lines in an empty text window:

```
NewDataWindow;
for i := 1 to nrRows do
  data[i,1] := 2 ^ i;
```

When you click the button "To Menu" or choose Add to Menu from the 'Customize' menu, pro Fit finds that you have entered the body of a program and that you have used the variable i. It therefore adds the necessary Pascal syntax and then compiles your program. Your complete program will look like this:

```
program User_Program;
var i;

begin
     NewDataWindow;
     for i := 1 to nrRows do
       data[i,1] := 2 ^ i;

end;
```
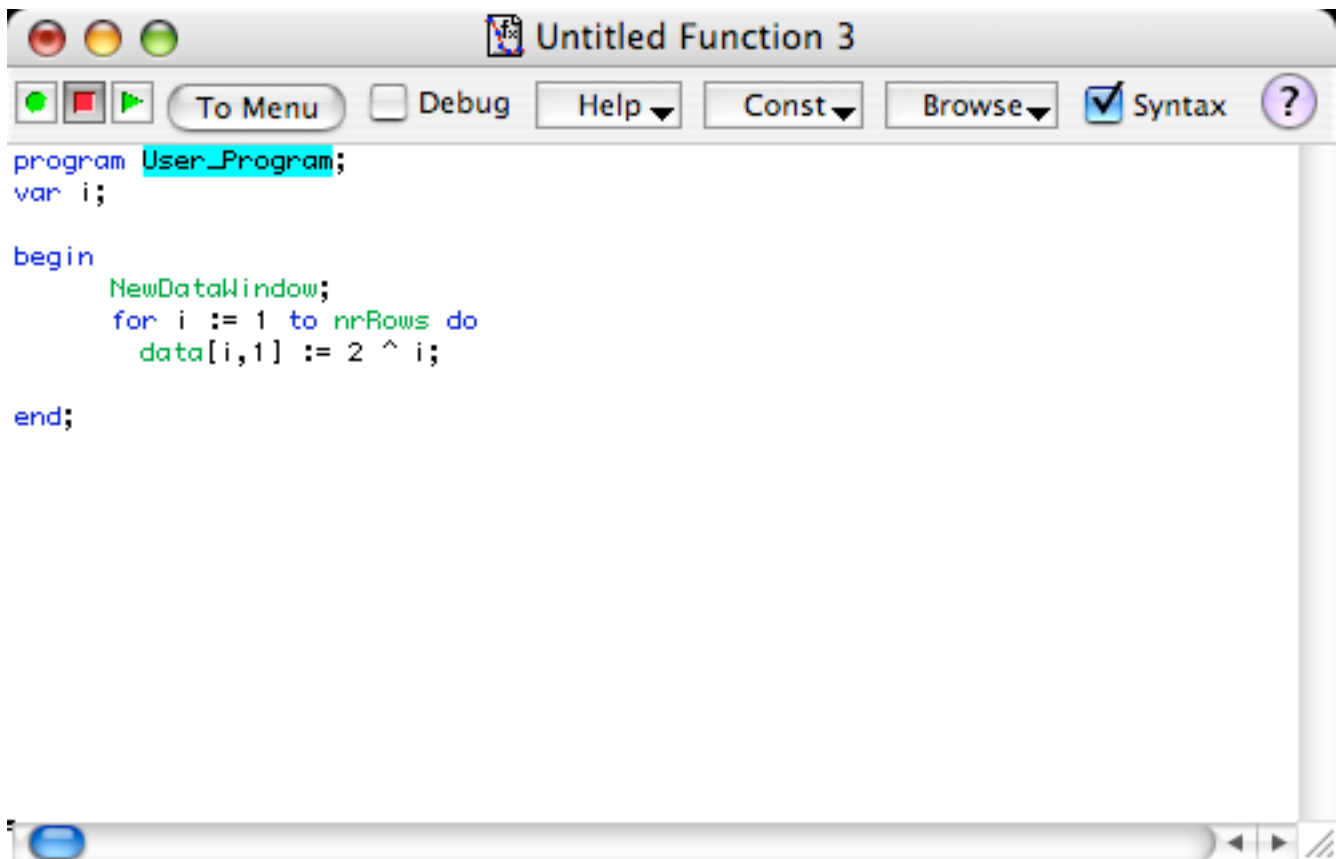
Functions with more than one output (multi-valued functions)

It is possible to define a function that has multiple output values. The 'default output value' can then be specified in the same way that you specify the default input value. Using the corresponding check box or using the Func menu. The case where a function has only one output is the simplest. Then the single output value can be set by assigning a value to the name of the function, or to a predefined variable called *y*. As an example,

```
function electricfield:real;
var
    E:real;
Begin
    E:=1/x;
    electricfield := E;
end;
```

or

```
function electricfield:real;
var
    E:real;
begin
    E:=1/x;
    y := E;
end;
```

The predefined variable *y* to which the output value is assigned becomes a predefined *array* for multi-valued functions. To tell pro Fit that a function has multiple output values there are two possibilities:

- Use the 'outputs' statement to declare and initialize the output values array. Example:

```
function vectorfield(phi:real):real;
outputs
      y[0]:=0, 'E (absolute value of the field)';
      y[1]:=0, 'Ex (x-component of the field)';
      y[2]:=0, 'Ey (y-component of the field)';
var
      E:real;
      begin
      E:=1/x;
      y[0]:=E;
      y[1]:=E*cos(phi);
      y[2]:=E*sin(phi);
end;
```

This allows to define, for all output values, the names with which they must be identified in the parameter window when output values are displayed, or in menus.

- Use var parameters, following the standard Pascal syntax, to declare additional output values:

```
function electricfield(phi:real; var Ex, Ey:real):real;
var
      E:real;
      begin
      E:=1/x;
      electricfield := E;
      Ex:=E*cos(phi);
      Ey:=E*sin(phi);
end;
```

Note that, following the pascal convention, the above function has three output values. The "usual" one specified by the function name plus the two additional ones specified as "var" parameters. The parameter window uses the names of the "var" parameters for the additional output values and the name "y" for the output value corresponding to the function name.

Instead of using the names defined in the function header, it is also possible to use the predefined array for the output values:

```
function electricfield(phi:real; var Ex, Ey:real):real;
var
   E:real;
begin
   E:=1/x;
   y[0]:=E;
   y[1]:=E*cos(phi);
   y[2]:=E*sin(phi);
end;
```

You can even mix the predefined output value array with the names you defined in the header, even inside the `outputs` statement:

```
function electricfield(phi:real; var Ex, Ey:real):real;
outputs
   y[0]:=0, 'E (absolute value of the field)';
   Ex:=0, 'Ex (x-component of the field)';
   Ey:=0, 'Ey (y-component of the field)';
var
   E:real;
begin
   E:=1/x;
   y[0]:=E;
   Ex:=E*cos(phi);
   Ey:=E*sin(phi);
end;
```

Selective calculation of output values

In most cases it is not necessary to calculate all output values of a function. The classic example is when only one of the output values is being plotted in a normal 2D plot. In order not do unnecessary calculations and to speed up things, pro Fit provides a predefined function – `output(i:integer)` – that you can call to find out if a given output value must be calculated. Note that sometimes a function should calculate more than one output value at the same time, as an example when the output values are displayed in the parameters window, or when a function is tabulated. The predefined function output accepts the index of the output value as a parameter and returns true if it must be calculated:

```
function electricfield(phi:real; var Ex, Ey:real):real;
outputs
   y[0]:=0, 'E (absolute value of the field)';
   Ex:=0, 'Ex (x-component of the field)';
   Ey:=0, 'Ey (y-component of the field)';
var
   E:real;
begin
   E:=1/x;
   if output(0) then y[0]:=E;
   if output(1) then Ex:=E*cos(phi);
   if output(2) then Ey:=E*sin(phi);
end;
```

Determining how multiple output values are rendered in the preview window

Input and output values of a function have various *properties* that can be set by the dedicated functions `SetParameterProperties` and `SetOutputProperties`. These functions can be called by any pro Fit program or function, and also by a function that wants to set some advanced properties of ist outputs. One of these properties determines if an output appears in the preview window even when it is not defined as the default output. To get an idea of what this means, look at one of the predefined Peaks functions in the preview window.

By using these calls in its initialization routine, a function can determine if and how its outputs must appear in the preview window, and even set their names there, instead of using the `outputs` statement:

```
function electricfield(phi:real; var Ex, Ey, r,h,v:real):real;
var
   E:real;
procedure Initialize;
begin
   SetFunctionProperties(function '', preview groupOutputValues);
   SetOutputProperties(output 0,
                       name 'E (absolute value of the field)',
                       outputGroup 1);
   SetOutputProperties(output 1,
                       name 'Ex (x-component of the field)',
                       outputGroup 1);
   SetOutputProperties(output 2,
                       name 'Ey (y-component of the field)',
                       outputGroup 1);
   SetOutputProperties(output 3,
                       name 'r (distance from the origin)',
                       outputGroup 2);
   SetOutputProperties(output 4,
                       name 'h (horizontal displacement)',
                       outputGroup 2);
   SetOutputProperties(output 5,
                       name 'v (vertical displacement)',
                       outputGroup 2);
end;

begin
   E := 1/x;
   y[0]:=E;
   Ex:=E*cos(phi);
   Ey:=E*sin(phi);
   r:=x;
   h:=x*cos(phi);
   v:=x*sin(phi);
end;
```

Here, the `SetFunctionProperties` call is used by the function to assign to itself the property that its outputs are shown in the preview window whenever they belong to the same *group* as the default output. The remaining `SetOutputProperties` calls give the outputs names that are used in the parameter window, and assign the outputs to two different *groups*. Ouputs from the same group are shown at the same time in the preview window.

On-line help for programming

The introduction given above only scratches the surface and omits to describe many of the possibilities that you have to determine how a function or program that you define behaves and can be used within pro Fit. An example of a more advanced feature has been given in the last section. Another example is the possibility of defining command-key equivalent for calling a program. Or you can also put a program in a submenu. To find out about all these possibilities and how to use them, the on-line help

provided by pro Fit is the place to go (you can also read on in this chapter but the on-line help is a more handy tool).

The help menus

When defining functions and programs, you can use a series of predefined names, functions and procedures. To help you use them, pro Fit provides a popup menu "Help" in the header of all function windows.
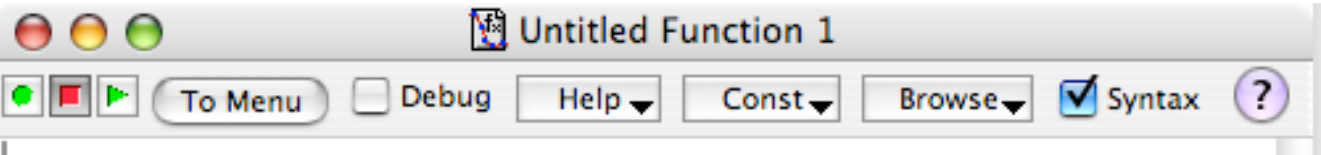
The "Help" popup menu lists all predefined routines, names, and syntax elements that you can use. The items are organized hierarchically. It is easy to find an item by moving the mouse over all the different headings.

In addition to this, the function window provides a popup menu called "Const" that provides a list of some of the most important constants of nature (thinks like the speed of light, the charge of an electron, the mass of a neutron, etc.).

When you move the mouse over an entry in the menus "Help" and "Const", a help tag is shown giving a short description. When you choose an entry and release the mouse, its definition is pasted into the function window.

You can enable/disable the help tags for these two menus by choosing the entry "Show Help Tags" from the popup menu "Help".

Browsing functions and programs

Navigating a lengthy function or program definition can be difficult. To get a quick overview of your definition, click the popup menu "Browse" in the toolbar of the function window.
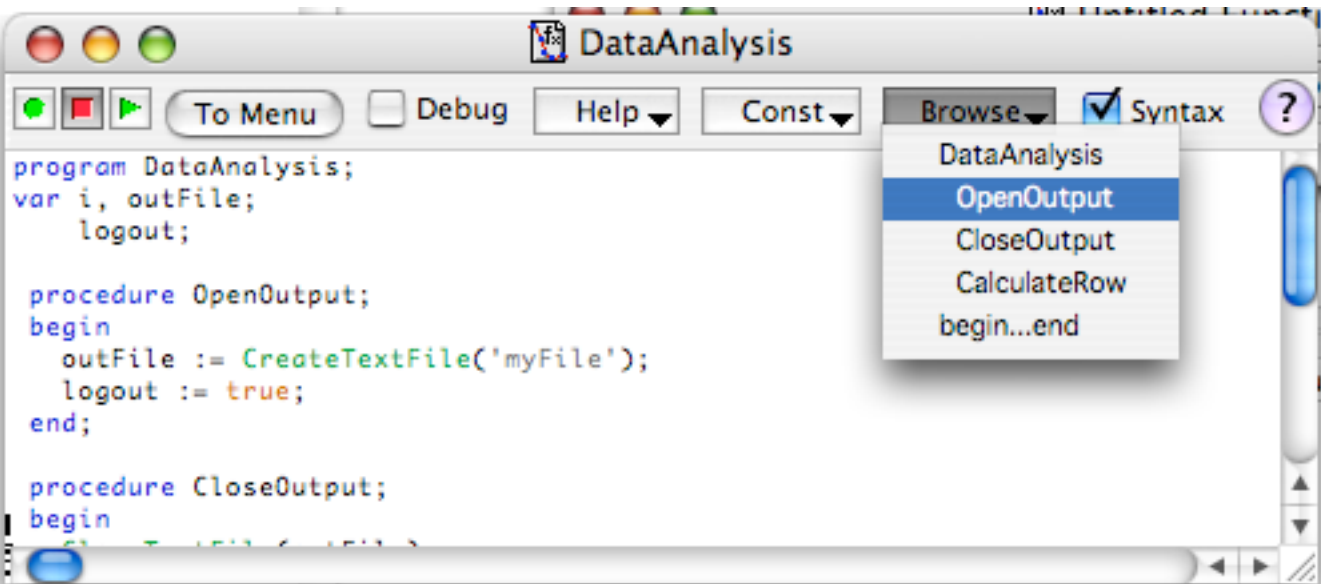
The menu shows a list of all functions, procedures and programs defined in the file. Choosing an entry from this list takes you there.

Finding the definition of a symbol

If you want to find the definition of a symbol, variable or command that appears in a function window, double-click it while holding down the option key. For example, if a function window looks as follows:

```
DataAnalysis

program DataAnalysis;
var i, outFile;
    logout;

  procedure OpenOutput;
  begin
    outFile := CreateTextFile('myFile');
    logout := true;
  end;

  procedure CloseOutput;
  begin
```
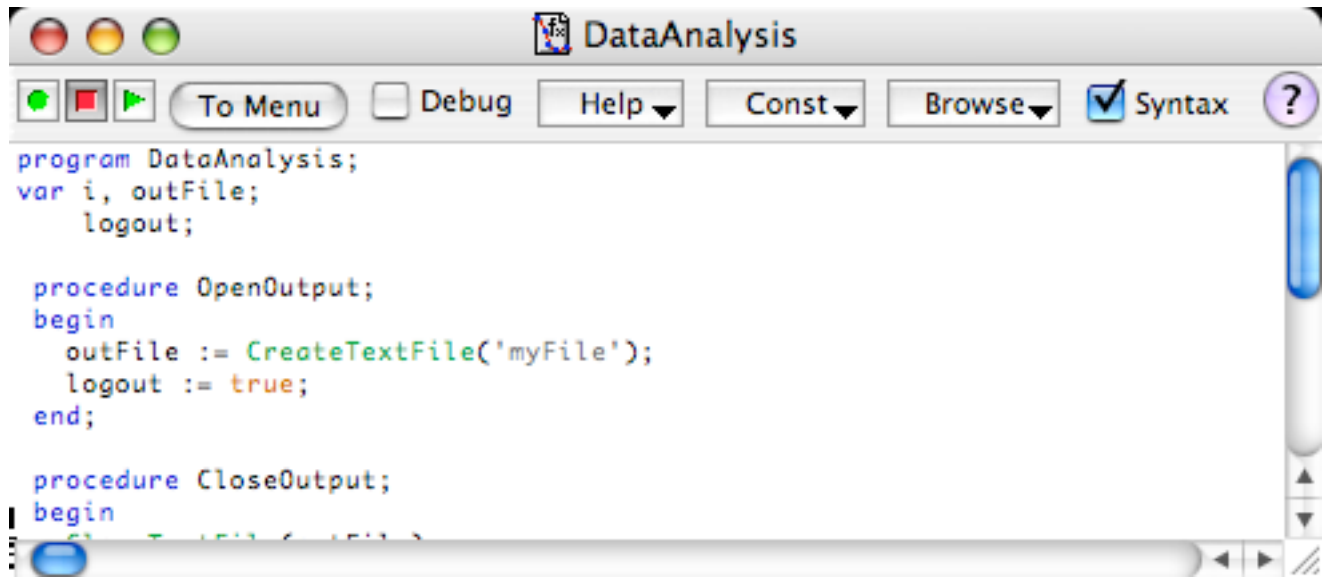
and you want to know how "CreateTextFile" is defined, double-clicking it while holding down the option key brings up its definition:

pro Fit Help

Q NewWindow

CreateTextFile

function CreateTextFile (fName:string, opt:integer): integer

Creates a text file and returns a reference number used to identify this file. Parameters:

fName: The name of the file. If fName is of the form '?' or '? :myname', a save dialog box is displayed. The text file will be created in the selected directory.

opt: Defines how to treat existing files. This parameter can take the following values:

0 = new file required,
1 = append if existing text file,
2 = overwrite if existing text file,
3 = overwrite any existing file

Related topics

WriteToTextFile, CloseTextFile

This window is part of pro Fiit's extensive on-line help, which helps in using pro Fit as well as in programming functions or data transform algorithms. This help is permanently accessible through pro Fit's Help menu, and you can browse it at will, or use its "search" to learn about how to use pro Fit and about the details of various commands.

Automatic Macro Recording

pro Fit can "record" most operations that you perform and generate a Pascal program or an Apple Script from them. (Open Apple's script editor to record your activity as an apple script. See chapter 11, Apple Script, for more information.)

If you do not know how to program a certain action with pro Fit's definition lagnuage, switch on recording, perform the action you want to program, and look at the recorded commands.

Each text window has record, play and stop buttons:
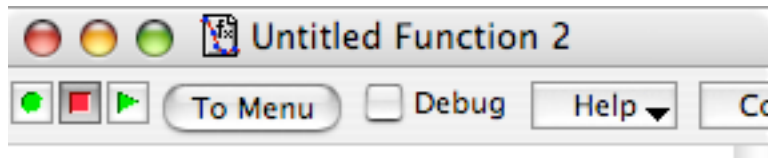
The record button is the one with the circle in its center, the stop button is the one with the square, the play button is the one with the triangle.

To record your actions, click the record button. pro Fit will automatically generate a Pascal script for nearly everything you do. When you have finished recording, click the stop button. Then you can replay what you did by clicking the play button.

Alternatively, you can use the commands "Start Recording", "Stop Recording" and "Run" (or "Run Selection") from the Customize menu.

If you only want to run a part of a script, first select it and then click the play button (or choose "Run Selection") from the Customize menu. If you don't select a part of the script before clicking the play button, then the whole script is run. If there are function or program definitions in the midst of the script, they will be added to pro Fit's menus.

The recorded commands appear at the current insertion point in the text window. You cannot edit the text window while it is recording.

You can record new commands at any place inside an existing program definition. Simply position the cursor where you want the new commands to appear, and click the "record" button.

Syntax of function and program definitions

This section gives a full description of the elements of pro Fit's syntax for function and program definitions.

Information on how to define programs is found under "program definition syntax". Information on how to define functions is found under "program definition syntax" *and* under "function definition syntax". You need to look under both headings because the function definition syntax is based on the program definition syntax. Read the sections devoted to programs to obtain an explanation of all the general features that are available to programs as well as functions.

Program definition syntax

The structure of a program definition is basically identical to that of a program in standard Pascal. It starts with the keyword `program` followed by the name of the program and a semicolon. Then you can optionally define some variables, constants, procedures or functions for your own use. The main part of the program (where the execution starts) is placed between `begin` and `end` at the end of the program.

`program myProg;`	the name of the program, myProg, will appear in the Prog menu.

`const c = 3e8;`
`var   u,v: real;`
`      done:boolean;`
	optional, definition of constants and variables.

`procedure MyProc;`
`begin`
`  statements...`
`end;`
	optional, definition of a local procedure or function used by the program.
	Note that you can call local procedures recursively.

`....`	more definitions of functions or procedures can follow here

`procedure Initialize;`
`begin`
`  statements...`
`end;`
	optional, the procedure Initialize that is called once when the program is compiled and added to the function menu.
	Any initialization of global variables can be done here.

`begin`
`  statements...`
`end;`
	the main body of the program where execution starts.
	Note the ';' after the end.

After the title of the program, you can define *constants* and *variables*.

The definition of *constants* is preceded by the keyword `const`, which is followed by the name of each constant, the operator '=' (not ':='), and the value of the constant. Example:

```
const c = 3e8;
      startValue = 22;
```

Once you have defined the value of a constant, you cannot change it anymore.

The definition of *variables* is preceded by the keyword `var`, which is followed by a list of variables.

```
var   u,v: real;
      done:boolean;
      m:matrix[3];
      c:complex;
```

Note that you can specify the type of each variable (such as `real`, `boolean`,`complex`). If you omit the type specification, it is assumed that the variable is of type `real`.

Variables and constants that you define in the head of a program can be accessed by all statements within the program and the program's procedures and functions.

You can use any name you like for a constant or variable (as long as it is not yet used for any other purpose). It can contain letters and digits but must start with a letter. Examples for names are:

```
myFunc, xx, J0
```
legal names

```
   2ToX                        illegal (starts with a digit)

   then                        illegal (reserved keyword)
```

The same rules apply to the names of procedures and functions (see below).

Following the definition of constants and variables, you can (optionally) define local procedures and functions. The general form of their definition is:

... for a procedure:

```
procedure MyProc(m,n:real; i: integer);
   variable and constant definitions ...
begin
   statements, separated by semicolons
end;
```

... for a function:

```
function MyFunc(m,n:real; i: integer):real;
   variable and constant definitions ...
begin
   statements, separated by semicolons;
   myFunc := return value
end;
```

In this case, `MyProc` (or `MyFunc`) is the name of the procedure (function). The name is followed by a list of arguments in brackets. If the procedure or function has no arguments, this list (including the brackets) is omitted. In our examples we have three arguments: `m,n` and `i` together with their type definitions. If you define a function, the declaration of its return type follows after the argument list. Then follows a semicolon.

After the line defining the name of the function or procedure you can define constants or variables using the same syntax as described for the program (see above). These items are only known within this procedure or function.

The statements of the procedure or function follow, enclosed by `begin` and `end`;

You can call a procedure or function anywhere after its declaration, like this:

```
...
MyProc(1.72,3.13,20);
r := MyFunc(1.71,3.14,10);
...
```

Local functions and procedures can also have var parameters. When you change a var paramter, you change the value of the corresponding variable of the calling function. Example:

```
program Test;
  procedure Increase(var a:Real);
    {increase value of a by 1}
  begin
    a := a+1;
  end;
begin
  k := 1;
  Increase(k); {increases k by 1}
  Writeln(k);  {writes 2}
end;
```

If you define a procedure having the name `Initialize`, it is called automatically whenever the program is added to the menu. Within `Initialize` you may want to initialize any variables or print some information into the Results window. Here is an example:

```
program DoMyStuff;
var inputColumn:integer;
        {where our data comes from}

procedure Initialize;
 {prints a description of the program and }
 {sets the default value of inputColumn }
begin
 Writeln('This program converts a data column');
 Writeln('into normalized units.');

 inputColumn:=3; {inititialization}
end; {of initialize}

begin  {main part of program}

{ask for an input column, default is the one}
{that was set in initialize}
 Input('which column?',inputColumn);

{transform data}
....
end; {of main part}
```

The above program uses the predefined function `Writeln` to output text to the results window and the function `Input` to ask the user for a column number. All predefined functions are described in pro Fit's on-line help.

Example

Let us look at an example of a fully functional program:

You have a data window that contains data in the first two columns. The first column contains positive and negative numbers. You are only interested in the positive numbers and you want to delete all rows which have a negative number in the first column.

Here is the program:

```
program EliminateNegatives;
var i:integer;

procedure DeleteRow(r:integer);
    {deletes the row r and shifts up}
    {all following rows}
var m,n:integer;
begin
  for n:=1 to 2 do
  begin
    for m:=r to nrRows-1 do
        if DataOK(m+1,n) then
           data[m,n] := data[m+1,n]
        else ClearData(m,n);
    ClearData(nrRows,n);    {clear last row}
  end; {of for loop}
end; {of deleteRow}

begin  {main part of program}
 i:=1;
 while i <= nrRows do
 begin
    if DataOK(i,1) then
      if data[i,1] < 0 then begin
        DeleteRow(i); i:=i-1;
      end;
    i:=i+1;
 end; {of while loop}
end; {of main part}
```

This program tests all numerical values in column 1. This is done in a **while** loop. A while loop has the general form

      while *condition* do statement;

Its statement is executed as long as its condition is true. If you have more than one statement in a while-loop, they must be enclosed by `begin` and `end`.

Our example program executes the while-loop for all rows in the data window (`while i <= nrRows`). If a data cell in column 1 and row i contains a negative number, the procedure `DeleteRow` is called, which deletes the row `i` by shifting all following rows up.

The procedure `DeleteRow` calls `ClearData(r,c)`, which is a built-in procedure of pro Fit. `ClearData(r,c)` removes any number from the cell in column `c` and row `r`.

In the examples above, we have used the 'for' loop and the 'while' loop. Let us summarize their use and introduce the third kind of loop (the 'repeat' loop):

Loops

pro Fit supports three kind of loops, two of which we have already seen (for-loops and while-loops). The third one is the repeat-loop. The loop statements are:

The while-loop

> `while` *condition* `do` *statement*`;`

The statement of the while-loop is executed as long as the expression in condition returns true. If more than one statement should be executed in the loop, the statements must be enclosed by `begin` and `end`.

The for-loop

> `for` *loopVariable* `:=` *startValue* `to` *endValue* `do`
> > *statement*;

A for-loop executes its statement for all integer values of its variable between startValue and endValue. If startValue equals endValue, the for-loop is executed only once. If the startValue is larger than the endValue, the for-loop is never executed. If more than one statement should be run in the loop, the statements must be enclosed by `begin` and `end`.

An alternative form of the for-loop is

> `for` *loopVariable* `:=` *startValue* `downto` *endValue* `do`
> > *statement*;

In this for-loop the value of the loop variable is decreased by one after each execution of the loop statement. The loop is terminated as soon as loopVariable < endValue.

The repeat-loop

The last kind of loop is the repeat-loop. Its general form is

> `repeat` *statement* `until` *condition*`;`

In contrast to the while-loop, the statement of a repeat loop is always executed at least once. After the execution of the statement, the condition is tested. If the condition is true, the loop is terminated, else the loop statement is executed again until the condition becomes true.

Loop control statements: cycle and leave

You can place the keyword **leave** into a for-, while- or repeat-loop to exit the loop even if its end-condition is not yet reached. Example:

```
for i := 1 to NrRows do
begin
 if not DataOK(i,1) then
 begin
   Writeln('Empty cell - loop aborted');
   leave;   { exits the for-loop }
 end;
 ....
end;
```

The above example loops through the first column of a data window and does some calculations (indicated by '`....`'). If, however, an empty cell is found, the loop is aborted.

You can place the keyword **cycle** into a for-, while- or repeat-loop to immediately start a new iteration of the loop. Example:

```
for i := 1 to NrRows do
begin
 if not DataOK(i,1) then
 begin
   Writeln('Empty cell skipped');
   cycle;   { goes to next value of i }
 end;
 ....
end;
```

The above example loops through the first column of a data window and does some calculations (indicated by '`....`'). If an empty cell is found, the calculations are skipped and the loop is continued with the next value of `i`.

Optional parameter lists

Usually, you pass parameters to procedures and functions using the standard Pascal syntax. For example, you write

```
DrawRect(10, 10, 50, 100);
```

In other words, you pass a value for each parameter and separate the parameters by commas.

However, some of pro Fit's predefined procedures use an "optional parameter list" for passing values, for instance

```
CloseWindow(window 'Data 1', saveOption dontSave);
```

In the above example, "window" and "saveOption" are the names and 'Data 1' and dontSave the values of the parameters that are passed to the procedure CloseWindow. In other words, each parameter has a name that must be passed in front of its value.

The advantage of this calling convention is that you can omit some parameters (if you want to use their default values). For example, you can call

```
CloseWindow(saveOption ask);
```

In this example, we have omitted the parameter "window" and use its default value (the front window) instead.

The pro Fit's on-line help state which of pro Fit's predefined procedures use optional parameter lists.

Aborting procedures, functions and programs

Use the keyword `Halt` to immediately end the execution of a function or program. Use the keyword `Exit` for exiting from a local function or procedure to the caller.

The following is an example of a program calculating the sum of the presently selected cells in a data window. The program aborts when the selection contains empty data cells. (Note that it uses the predefined variables `selectLeft, selectRight, selectTop, selectBottom` which return the enclosing rectangle of the currently selected data cells.)

```
program CalcSum;
var row, col: integer; sum: real;
begin
  sum := 0;
  for col := selectLeft to selectRight do
   for row := selectTop to selectBottom do
   begin
    if not DataOK(row,col) then Halt;
    sum := sum+data[row,col];
   end;
  writeln(sum);
end;
```

The following program does basically the same as the one above, but the sum is calculated in a local function, which is aborted by `Exit`:

```
program CalcSum;

 function SumSelection:real;
 {sums the selected data, returns}
 {-1 if a selected cell is empty}
   var row, col: integer; sum: real;
 begin
  sum := 0;
  for col := selectLeft to selectRight do
   for row := selectTop to selectBottom do
   begin
    if not DataOK(row,col) then begin
        SumSelection := -1;
        Exit;
      end;
    sum := sum+data[row,col];
   end;
  SumSelection := sum;
 end;

begin
  Writeln(SumSelection);
end;
```

Note: Calling `Exit` from the main body of a function or program has the same effect as calling `Halt`.

Predefined constants, functions, procedures, and operators

This section lists the operators and the most important predefined constants that are available in the definition syntax. An full list of all predefined functions, procedures and constants is found in pro Fit's on-line help.

The following are the most important predefined constants:

```
π (or pi)              = 3.141592...
true                   = 1
false                  = 0
INF                    infinity (1/INF=0)
```

The operators are identical to those that are defined in standard Pascal. In addition, the power operator (`**` or `^`) has been added. The operators – in ascending order of precedence – are:

`= <> <= < > >=`	comparison, returning true (1) or false (0)
`+ - or`	add, subtract, logical 'or'
`* / and`	multiply, divide, logical 'and'
`**, ^`	power (x `**` y $=$ x^y $= x^y$)
`not`	logical 'not'

You can change the order of precedence of the operators in the above list by using brackets: '(' and ')'.

Note that there are two ways for using the power operator (x**y and x^y). They are equivalent. Use whichever you prefer.

> On some machines, `x**y = x^y` is calculated as $\exp(y \ln(x))$. As a consequence of this, the `x^y` may not work for negative x and may be slow. Therefore, you should not use this notation for calculating small integer powers (for example: use `sqr(x)` instead of `x**2`).

Note for Pascal programmers: `^` is used for the power operator. pro Fit does not know anything about pointers and `^` is not used for dereferencing.

> The order of precedence for the operators is the same as in standard Pascal. But since the pro Fit definition language does not distinguish between boolean and real expressions (refer to the next chapter), this order of precedence provides a dangerous pitfall
>
> `a>x and b>y`   will be compiled as   `(a > (x and b)) > y` !!
>
> Use brackets to clarify what you want:
>
> `(a>x) and (b>y)`

Note: In contrast to some other programming languages, *all* the expressions in a composite logical expression of the form

(*condition* 1) `and` (*condition* 2) `and` (*condition* 3)
will be evaluated, even if *condition* 1 returns false.

Function definition syntax

If you want to define a function of your own to use it for fitting or plotting, you must write a **function definition**. The structure of a function definition is the same as the structure of a program definition, but it can optionally contain additional information about the parameters and the contents of the parameters window. This additional information is placed right at the beginning of the function definition.

A function definition starts with the keyword `function` instead of `program`. Then follows (optional) information on the parameters and the parameters window:

`function myFunc;`	the name of the function, myFunc, will appear in the Func-menu.
function myFunc(ampl , freq: real; var out1, out2: real);	*optional*, definition of names that will be used to access input and/or output values in the function code and as a default name in the parameters window. Output values are preceded by the keyword 'var'.
`description    'text1','text2';`	*optional*, these two strings will appear in the parameters window.
`parameters 4;`	*optional*, the number of additional (besides the standard x-value) input values (max. 128)

`inputs`
`a[1]:=1.2,active;`
`a[2]:=3.0,inactive,'name';`
`a[3]:=2.0,constant;`
`a[4]:=1,active,'i',0,INF;`
 	
`inputs`
`ampl:=1.2,active;`
`freq:=3.0,constant;`
	optional, the default values for the parameters, their default mode, parameter-window name, lower and upper limit (see the Chapter 8, *Fitting*). If you do not define the defaults for a parameter it will be 0, inactive and limited by -INF and INF. If you do not define a parameter-window name for a parameter its default name will be used. The default name is either the name you define in the function header (e.g. 'ampl') or 'a[i]'.

`outputs`
`  y[0]:=0, 'E (absolute value)';`
`  Out1:=0, 'Ex (x-component)';`
`  Out2:=0, 'Ey (y-component)';`
	optional, the default values for the outputs and their name to appear in the parameter window. If you do not define the defaults for an output value, its value will by default be 0.
	Alternatively, if you do not want to define default values and names for all outputs of a function, but you merely want to state that your function has n outputs, simply assign the constant n to the outputs keyword, e.g.
	`  outputs = 3;`

`const` `  c = 2.997E8;`	*optional*, the definition of constants as in standard Pascal.

`var`
`  temp: extended;`
`  myVar,t: integer;`
	optional, variable declarations as in standard Pascal.

After this, you can (optionally) define your own local procedures and functions.

Then follows the "body" of the function definition between `begin` and `end`. In this body, you must calculate the function's *y*-value from its *x*-value and its parameters. For this, you can use the following variables:

x	The primary input value variable of the function
a[1] ... a[n]	The remaining input values (parameters) of the function. Up to 128 input values can be used.
y	The output variable, the function's return value, for single-values functions. It must be set by your function.
Y[0] ... y[n]	An array holding the output values of a multi-valued function. y[0] is the function's default output value.

It is possible to define your own input and output names in the function header and to use your own names instead of the `a[1]...a[n]`:

```
function foo(ampl, freq, phase: real; var result);
begin
    result := ampl*cos(freq * x +phase);
end;
```

If you do this, input and output values retain their numbering, defined by their sequence when you define them (`ampl, freq, phase`). The `a[i]` remain available as synonims (`a[1]=ampl, a[2]=freq, a[3]=phase`) and the indices can still be used in predefined function such as `SetParamName`.

Example 1:

You want to define the function:

$$y = a_1 \ln(a_2\, x^2)$$

Your definition looks like this:

```
function logSquare(K, Q: real);
begin
  y := K*sqrt(Q*cosh(x));
end;
```

This is a function in its most simple form. If you work with it often, you may want to assign default values to the parameters. You will also see that Q should not be negative. You might therefore improve the above definition as follows:

```
function logSquare(K, Q: real);
inputs
 K := 1, active, 'K (amplitude)';
 Q := 1, active, 'Q (multiplier of cosh)', 0, INF;
begin
   y := K*sqrt(Q*cosh(x));
end;
```
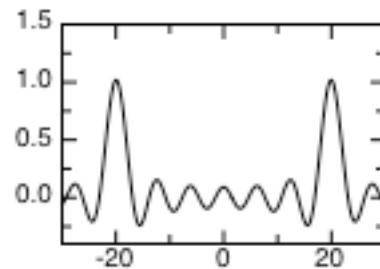
The first line after the keyword `inputs` defines the default value, default mode (active means that it will be varied in a fit) and the name of *K* that will appear in the parameter window. The second lines defines the default value, mode and name as well as the lower and upper limit of Q .

Example 2:

You want to define the function

$$y = a_1 \operatorname{sinc}(x-x_1) + a_2 \operatorname{sinc}(x-x_2) ,$$

with $\operatorname{sinc}(x) = \sin(x)/x$.

The value of the function sinc is not defined for *x*=0, but it converges to 1 for $x \varnothing 0$. When calculating sinc, we must test if its argument is 0 to handle this special case.

Since the sinc function is used twice in our example, it makes sense to put it into a local function.

```
function DoubleSinc;

inputs a[1] := 1,active,'a1';
        a[2] := -20,active,'x1';
        a[3] := 1,active,'a2';
        a[4] := 20,active,'x2';

function Sinc(u:real):real;   { sin(u)/u }
begin
  if u=0 then sinc:=1         {0/0 is illegal}
  else sinc:=sin(u)/u;
end;

begin  {"body"}
  y := a[1]*sinc(x-a[2]) + a[3]*sinc(x-a[4]);
end;
```

Special procedures in a function definition

As in a program, the procedures that you define within a function definition can have any valid name you want. However, there are some reserved names for special procedures (`Initialize`, `Check`, `First`, `Derivatives`, `Last`) that you can define to customize and optimize your function definition. These procedures are called to perform special actions. For example, one of them (`Derivatives`) is called to calculate your function's partial derivatives. Another (`Check`) can check a value that was entered into the parameters window.

The following describes these special procedures. A summary is provided at the end of the section.

Function Check

This procedure is only used to include some advanced features in your function. It can make function definitions more user-friendly. `Check` is called each time the user changes an input value in the parameters window. It can check the input value that was changed and act accordingly. For example, it can refuse a value if it is not acceptable. It can also recalculate some other input values and cause the parameters window to be redrawn. `Check` can use the following predefined variables and constants:

`pNumber`	The number of the modified parameter
`a[1] .. a[n]`	The input values as they appear in the parameters window. They can be checked and/or changed.
`mode[1] .. mode[n]`	The mode of each input, which can be `active`, `inactive` or `constant`. You can check and/or change the modes.
`active, inactive, constant`	These three constants can be used to be compared to or assigned to `mode[i]`.
`check`	The function must store its return value in this variable.
`ok, bad, update`	One of these three constants must be returned in the variable check.

`Check` must return one of the values `ok`, `bad` or `update` in the variable `check` to tell pro Fit if it should accept the new input value and what it should do with the parameters window:

- If `check=ok`, pro Fit accepts the new parameter.
- If `check=bad`, pro Fit refuses the new parameter and shows the old one in the parameters window.
- If `check=update`, pro Fit accepts the new parameter and redraws (updates) the whole parameters window. Use this feature whenever you have changed a parameter other than `a[PNumber]` in the function `check`, so that the user can see these changes.

For example your function can have two input values that represent the same value in two different units of measurement. `Check` can be used to update the value of one input value when the other input value is changed.

Note for advanced users: `Check` is not called during fitting. It is called once when fitting is complete. Don't use `Check` for calculating intermediate results for later use in the evaluation of the function. You won't notice anything wrong as long as you modify the values in the parameters window, but your function will not work when fitting. Always use the procedure `First` (see below) for calculating intermediate results.

Procedure Initialize

This procedure is used for advanced programming. It is called exactly once after compilation of your function or program. You can use this procedure to initialize the value of variables or to write some instructions into the Results window.

Procedure Derivatives

This procedure is optional. If defined, it is used during fitting with the Levenberg-Marquardt algorithm. This algorithm uses the partial derivatives of the function with respect to its parameters. If you do not define the procedure `Derivatives`, the derivatives are calculated numerically, but this slows down the fitting process considerably. If you notice that fitting is particularly slow, you should define this function and at least calculate some derivatives (pro Fit will still calculate numerically any derivative you don't define). The procedure derivatives can use the following predefined variables:

`x`	The x-variable, the function's x-value
`a[1] .. a[n]`	The input values of the function.
`dyda[1] .. dyda[n]`	The partial derivatives. Must be set to dyda[i] := $\partial f(x)/\partial a[i]$ for all parameters that are not declared as *constant*.

`Derivatives` can set the values `dyda[i]` for some or all of your function's parameters. If you don't set a value, it will be calculated numerically.

Whenever a function is used by pro Fit, a call to the procedure `Derivatives` is always preceded by a call to the main part of the function. Therefore you may use temporary results from the main part of the function by storing them into global variables. This decreases the number of calculations your function must perform and makes fitting faster.

Example: You want to fit the function y = $a_1 \cdot \sinh(x)$, the partial derivative of which is $\partial y/\partial a_1$ = $\sinh(x)$. Calculating $\sinh(x)$ can take a lot of time, especially when you are working on a slow computer. To avoid calculating expressions twice, you can save temporary results in the main part of the function to use them later in the procedure `Derivatives`:

```
function MySinh;
var t: real;

procedure Derivatives;
begin
  dyda[1] := t;{ use t calculated in body}
end;

begin           {the function's body}
  t := sinh(x);    {save sinh for derivatives}
  y := a[1]* t;
end;
```

Procedure First

This procedure is used for advanced programming. It is called whenever the input values of a function have been changed – before the body (main part) of the function is called. The body of a function will never be called without `first` having been called beforehand.

The procedure `first` can use the following variables:

> `a[1] .. a[n]`   The parameters of the function.

The procedure `First` is mainly used for accelerating calculations that do not depend on the input value x. This can make a fit considerably faster. `First` should calculate all expressions that appear in a function but that do not depend on *x*:

> To calculate the mean deviation χ_R during fitting, pro Fit calculates the function for each data point (x_i, y_i). This may involve up to several thousand executions of the body of the function definition. If your function definition contains expressions that do not depend on the value of *x* (such as sin(a[2]–a[3])), they will still be recalculated for each new value of *x*, wasting a lot of time. You can evaluate these expressions in the procedure `First` and store their values in variables used by the main part of the function.

> `First` only works as described when the default input value of the function is the standard x-value. If you set any other input to be the standard input value, then pro Fit will call `First` every single time the body of the function needs to be calculated. Always use the standard x-value as the default input whever you need to optimize the speed of a function using the possibilities provided by `First`.

Another use of the procedure `First` is to perform some task before pro Fit starts to use a function. This is less common for functions defined inside pro Fit but it is often used when defining plug-ins (see Chapter 10) that need to allocate and deallocate memory only used while a function is running. The following is a small example of this particular use of `First` that also demonstrates a possible use of the procedure `last`:

```
function Foo;
var    firstTime:  boolean;
       data11:     extended;
       sinDiff:    extended;
       multiplier: extended;
procedure Initialize;
begin
    firstTime:=true; {initialize to true}
end;


procedure First;
begin
    if firstTime then
    begin   {the statements in this block are }
        {executed only once, before any other}
        {function call.}
        firstTime:=false;
        data11:=data[1,1];
        {perform here other calculations that}
        {do not depend on parameter values}
        {and do not depend on x}
    end;
    sinDiff:=sin(a[2]-a[3]);
    multiplier:=data11*a[1];
    {perform here other calculations that do not}
    {depend on x but depend on the parameter}
    {values.}
end;


procedure Last;
begin                   {finished using function.}
   firstTime:=true; {reset firstTime to true }
end;


begin           {the main part of the function.}
  y := multiplier * sin(x)/sinDiff;
end;
```

The above example uses the procedure `Last`:

Procedure Last

This is also a procedure used for advanced programming. It is called when all calculations, fitting, etc. are completed. It is the last piece of function code called by pro Fit before returning control to the user. `Last` can be used to clean up, to make final calculations, or to re-initialize some variables to their starting values, as is shown in the example above. Last can also be used to print some special messages or results in the results window or to alert the user of some event. For example, you can let your machine beep when fitting is finished:

```
procedure Last;
begin
  beep;
end;
```

Summary

The following table summarizes the special procedures listed above:

name	called when	predefined variables and constants
Check	whenever parameters are changed by user	pNumber a[1] .. a[n] mode[1] .. mode[n] active. inactive, constant check, ok, bad, update
Initialize	once after compilation	a[1] .. a[n]
First	whenever input values are changed (e. g. during calculations)	a[1] .. a[n]
Derivatives	during fitting, after calling the function's main part	x, a[1] .. a[n] dyda[1] .. dyda[n]
Last	when calculations are through	a[1] .. a[n]
function's main part	during fitting and other calculations	x, y, a[1] .. a[n]

Note that in addition to the specially predefined variables and constants, all procedures (as well as the function's main part) can use the general predefined variables, constants, functions and procedures listed in pro Fit's on-line help.

General comments about programming

Types

The following basic types are supported:

`real`	real is the standard type for floating point numbers. It has at least 64Bit accuracy. (optional, but equivalent type: extended)
`integer`	integer is the standard type for integer numbers. It has at least 32Bit accuracy. (optional, but equivalent type: longint)
`string`	string is the type for strings. It is at maximum 255 Bytes long.
`char`	char is the type for single characters.
`boolean`	boolean is the type for booleans. It takes the values `true` or `false`.
`complex`	complex is the type for complex numbers. It consists of two real values, the real and the imaginary part.
`vector[`n`]`	vector$n$] is the type for a vector with n complex elements. $2 \le n \le 4$. The i-th element of a vector v can be accessed using v[i]
`matrix[`n`]`	matrix[n] is the type for matrices with nxn complex elements. $2 \le n \le 4$. The element in the i-th row and the j-th column of a matrix m can be accessed using m[i,j]

Note: pro Fit 6 does not distinguish between real, integer and Boolean types. All these types are implemented as 8 byte floating point numbers.

1. Simple numeric types:

The boolean value *true* is represented by the real value 1.0 and *false* by 0.0. All non-zero values are interpreted as true in a boolean expression.

Most Pascal compilers on the Macintosh distinguish between the floating point types **extended**, **double** and **real**, which have different accuracy. All simple number types of the pro Fit definition language have extended accuracy. The accuracy and range of numerical values in pro Fit is given in Appendix C.

2. Complex type:

The **Complex** data type is used to represent complex floating point values having a real and an imaginary part. Example:

```
program ComplexTest;
  var c: Complex;
begin
  c := -1;
  writeln(sqrt(c));
end;
```

The above program recognizes that sqrt is called with a complex argument. Therefore, a complex version of the square root function is used, which can handle sqrt(-1). The output of the above program is:

```
  0.000 + i * 1.000
```

Type conversion from real (or other simple numeric types) to complex is automatic. For converting complex numbers to real, use one of pro Fit's predefined functions, such as `abs, phase, re, im` (see below). To define complex numbers, use the predefined function `compl` or the predifined constant `ii`, which fulfills `sqr(ii)=-1`.

All predefined functions in pro Fit, such as `sin, cos, gamma, erf,` etc. automatically become complex valued functions if they notice that their argument is a complex number, and return complex numbers as a result.

pro Fit expressions of the `complex` type can be used with all mathematical operators and with all mathematical functions. When the type of a parameter is `complex`, the function will recognize it and return an appropriate complex or real result. The following are the few special functions that only make sense for complex numbers.

`conj`	returns the complex conjugate of a complex number
`Re, Im`	return the real and imaginary part, respectively, of a complex number
`phase`	returns the phase ϕ of a complex number $r\,e^{i\phi}$
`abs`	returns the absolute value rof a complex number $r\,e^{i\phi}$
`compl`	used to define a complex number. compl(x,y) = x + i y

2. Matrix and Vector types:

The **Matrix** and **Vector** data types are used to represent 2 dimensional and 1 dimensional arrangements of complex floating point values.

```
program MatrixTest;
  var m: matrix[2];
begin
  m := matr2(1,2,ii,-ii);
  writeln(sqr(m));
end;
```

The above program recognizes that sqr is called with a matrix argument. Therefore, a matrix version of the square function is used. The output of the above program is:

```
{{1.00 + i * 2.00,2.00 - i * 2.00},{1.00 + i * 1.00,-1.00 + i * 2.00}}
```

Type conversion from real or complex to matrix or vector. To define matrices, use the predefined function `matr2, matr3, matr 4`.

All mathematical calculations will autmoatically recognize matrix and vector types, and interpret them correctly when it makes sense.

The following are some special functions to be used on vectors and matrices:

determinant	returns the determinant of a matrix
transp	returns the transposed matrix
adjoint	returns the adjoint matrix
outer	returns the matrix defined as the outer product of two vectors.
matr2,matr3,matr4	used to define a matrix, these routines take 4,9, or 16 complex parameters, respectively.
vect2,vect3,vect4	used to define a vector, these routines take 2,3, or 4 complex parameters, respectively.

pro Fit expressions of the `matrix` or `vector` type can be used with normal mathematical operators and functions when it makes sense. Mathematical operations between matrices and matrices, matrices and vectors, vectors and vectors, and matrices/vectors with numbers do the expected thing. In the table below, "m" stands for any `matrix[n]` type, "v" for any `vector[n]` type, and "c" stands for any `complex` or `real` number.

`m*m`	matrix multiplication, result is a matrix.
`m*v`	matrix times vector, both must have the same dimension, result is a vector
`m*c, v*c`	multiplication by a scalar. Every matrix or vector element is multiplied by `c`.
`1/m`	this is the inverse of the matrix `m`. Produces a run-time error if the matrix cannot be inverted. `1/m = adjoint(m)/determinant(m)`
`m1/m2`	matrix division. `m1` is multiplied with the inverse of `m2`.
`v1*v2`	scalar product between two vectors. The result is a number.
`abs(v)`	the absolut value of a vector. The result is a real number.
`sqr(v),sqr(m)`	translates to v*v, and m*m, respectively
`conj(v), conj(m)`	the complex conjugate is obtained by taking the complex conjugate of each individual element.
`compl`	used to define a complex number. compl(x,y) = x + i y

The following is an example of a program doing some matrix and vector calculations:

```
program SomeMatrixAndVectorCalculations;
var    m1,m2:  matrix[2];                         {two 2x2 matrices}
       mm,mm2: matrix[3];                         {two 3x3 matrices}
       v1,v2:  vector[2];                         {two vectors of length 2}
       c:      complex;                           {a complex number}
begin
   mm1:=matr3(1+ii*2,2*ii,3,4,5,6,7,8,9);         {define the elements of the 3x3 matrix mm1}
   mm2:=1/mm1;                                    {mm2 is now the inverse of mm1}
   m1:=matr1(1,2,3,4);                            {define the 2x2 matrix m1}
   m2:=sqr(m1*4.2)+3.3;                           {m2 is calculated from m1}
   v1:=vect2(1,2+ii);                             {define the vector v1}
   v2:=m2*v1;                                     {matrix multiplication of v1 gives v2}
   c:=v1*v2;                                      {c is the scalar (dot) product of v1 and v2}
end;
```

4. String and char types:

Use the type **Char** for representing simple characters, **String** for representing strings of up to 255 characters. Example:

```
program StringAndCharTest;
  var c: Char;
      s: String;
begin
  c := 'x';
  s := 'hi there';
  writeln(c); {writes "c"}
  writeln(s); {writes "hi there"}
  s := s + ', Joe'; {s now is "hi there, Joe"}
  c := s[2];  {c now is "i"}
end;
```

Conversion between Strings and Chars is automatic. For conversion between Char (ASCII values) and Integer use the functions `Ord` and `Chr`. For conversions between Strings and numbers, use `NumberToString` and `StringToNumber`.

To access the n-th character in a string `s`, use `s[n]`. In other words, strings are arrays of type char.

The following is a list of the most important functions for working with strings:

`Length`	Returns the length of a string.
`Pos, Delete`	Find/ delete a sub-pattern in a string
`UpperString, LowerString`	Convert between upper and lower case strings.

See pro Fit's on-line help for a complete list.

Arrays

pro Fit allows the definition of one-dimensional arrays. The following syntax is used:

    var *name*: array[*minIndex..maxIndex*] of *type*;

Where *name* is the name of the array, *minIndex* is its minimum index, *maxIndex* is its maximum index, *type* its type. Since types are ignored by pro Fit, you can omit "of *type*" in the declaration.

To access an array, use the syntax:

    name[*index*]

Example:

```
 vararr1: array[1..10] of real;
     arr2: array[0..100];
        i
 ...
  for i := 1 to 10 do arr1[i] := 0;
  arr2[33] := 22.1;
```

Note: the maximum size of all variables in a variable list is limited to 32 kBytes. This limits the size of an array to about 2700 entries for the FPU version of pro Fit, to about 3200 entries for the non-FPU version, and to about 4000 entries for the Power Macintosh version.

Multi-dimensional arrays are not supported.

Note that arrays are a general purpose object, and should not be confused with the built-in vector types that only support vectors of length 2, 3, and 4, and that are mainly used in conjuction with the matrix types to perform matrix and vector operations.

Bit operations

`BitAnd, BitOr, BitXor, BitNot`	logical bitwise operations: and, or, exclusive or, not
`BitShift, BitClr, BitSet, BitTst`	functions to handle bit-arrays

Data processing

`Statistics`	run statistical analysis, get results
`Sort, ReduceData, BinData`	sort, smooth or reduce data or prepare data for histograms
`FFT, InverseFFT`	FFT and inverse FFT
`Transform`	general data transfomations
`Transpose`	transposing rows and columns

Accessing the data window

`data[i,j],`
`DataOK,`
`ClearData,`
`TestData*, SetData*, GetData*`
	an array and some routines for accessing the data in the current data window

`GetCell, SetCell`	setting and reading cell contents, including text-cells.

`SetDefaultCols,`
`SetDataWindowProperties,`
`GetDataWindowProperty`
	set the default x, y, Δx, and Δy columns and the number of columns and rows in the current drawing window.

`xColumn, yColumn, xErrColumn,` `yErrColumn`	the column numbers of the x, y, Δx, and Δy columns in the current data window

`NrCols, NrRows,`
`SelectLeft, SelectTop,`
`SelectRight,`
`SelectBottom`
`GetSelection*`
	information on the selection area and the size of the current data window

`SelectCell, SelectRow,`
`SelectColumn, RowSelected,`
`CellSelected`
	set the selection and check if a single cell or a row is part of a (possibly discontinuous) selection.

`SetColumnProperties,`
`GetColumnProperty,`
`GetColName, SetColName,`
`GetColType, SetColType,`
`SetColWidth, ColEmpty`
	obtain and write titles of single columns and other column characteristics.

`GetDefaultData*,`
`GetColHandle*,`
`SetColHandle*`
	obtain column data in a single step from external modules.

All the above calls access the current data window. By default, the current data window is the frontmost data window. You can make another data window the current data window by calling `SetCurrentWindow(windowID)` with `windowID` being the window ID of the desired data window.

Input and output

`Input, SetBoxTitle`	displays a dialog for entering numerical values
`Ask, Alert`	show alert boxes
`Write, Writeln`	these procedures write into the results window

`CreateTextFile,`
`CloseTextFile,`
`WriteToTextFile`
	open and close text files, and redirect the output of the write, writeln functions to a text file.

Drawing

`SetLineStyle, SetLineColor, SetFillColor, SetFillPattern, SetDataPointStyle, SetBGDataPointStyle SetArrowStyle, SetTextStyle`	set the style of future drawing calls
`MoveTo, LineTo, Move, Line, DrawLine`	produce line drawings in the drawing window.
`OpenPoly, ClosePoly`	collect line-drawing calls to define a polygon
`DrawLine,DrawDataPoint, DrawPICT*, DrawRect, DrawEllipse, DrawArc, DrawText, DrawNumber`	create single drawing objects in the current drawing window.
`GroupBegin, GroupEnd`	group drawing objects.
`DisableDrawingUpdates`	inhibit updates in the current drawing window until a program is finished.
`GetSelectionBounds`	find the rectangle corresponding to the boundaries of the current selection in the current drawing window
`GetClickedCoord`	find the last clicked point in the current drawing window.
`NewShape, DeleteShape, ShapeExists, SelectShape, SetShapeProperties, GetShapeProperty`	generic routines for creating, checking, changing and deleting any type of shape in a drawing window. (Every object in a drawing window, e.g. rectangles, groups, graphs, legends, etc, is a shape.)

Unless explicitly mentioned in the definition, all the above calls access the current drawing window. By default, the current drawing window is the frontmost drawing window. You can make another drawing window the current drawing window by calling `SetCurrentWindow(windowID)` with `windowID` being the window ID of the desired drawing window.

The drawing routines work on a coordinate system that has its origin on the top left of the paper. Units are points (1/72 of an inch).
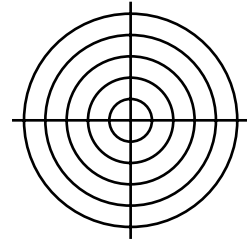
The following program creates a "bull's eye" at the point where you last clicked in the drawing window:

```
program BullsEye;
 const radius = 40; step = 8;
 var x0, y0, t:real;
      begin
 GetClickedCoord(x0, y0);
 GroupBegin;
 t := step;
 while t<=radius do
```

```
begin
  DrawEllipse(x0-t,y0-t,x0+t,y0+t);
  t := t+step;
end;
MoveTo(x0-radius*1.1, y0);
LineTo(x0+radius*1.1, y0);
MoveTo(x0, y0-radius*1.1);
LineTo(x0, y0+radius*1.1);
GroupEnd;
end;
```

The drawing routines accept floating point numbers as parameters. pro Fit uses a precise floating point coordinate system for drawings, and drawings created from a program will print at the highest resolution on all output devices.

Plotting in a graph

`PlotData, PlotFunction`	plot a data set or a function.
`SetLineStyle, SetLineColor, SetFillColor, SetFillPattern, SetDataPointStyle, SetBGDataPointStyle`	set the line style (line thickness, color...) of future line-plots and the style of future data points.
`SetCurveFill, SetEBarStyle`	set the filling options of plots and the appearance of error bars for the next curve or data set added to the current graph.
`OpenCurve, CloseCurve, OpenDataSet, CloseDataSet`	start/end the definition of curves or data sets for the current graph
`AddDataPoint, DrawDataPoint`	add a data point (possibly including error bars) to the current data set.
`MoveTo, LineTo, Move, Line`	define a curve in the current graph.

All the above calls access the current graph. To make a graph the current graph, double-click it while holding the command key down. From a program, you can use the call `SetCurrentGraph` to make a graph the current graph.

The following is a small example program drawing a Lissajous figure in the drawing window:

```
program Lissajous;
var xmin, xmax, ymin, ymax;
    centerH, centerV, {center of the figure}
    radiusH, radiusV; {and its radius}
    angle;
begin
  xmin:=1;xmax:=3;
  ymin:=2;ymax:=5;
  CreateNewGraph(xmin,xmax, ymin,ymax, false,false);
  centerH := (xmin+xmax) / 2;
  centerV := (ymin+ymax) / 2;
  radiusH := (xmax-xmin) * 0.4;
  radiusV := (ymax-ymin) * 0.4;
  SetLineStyle(1,2);

  OpenCurve('Circle');
  MoveTo(radiusH+centerH, centerV);
  angle := 0;
  while (angle <= 2*pi) do
  begin
    LineTo(radiusH*cos(3*angle)+centerH, radiusV*sin(2*angle)+centerV);
    angle := angle + pi/40;
  end;
  CloseCurve;

  SetLineStyle(1,1);
end;
```
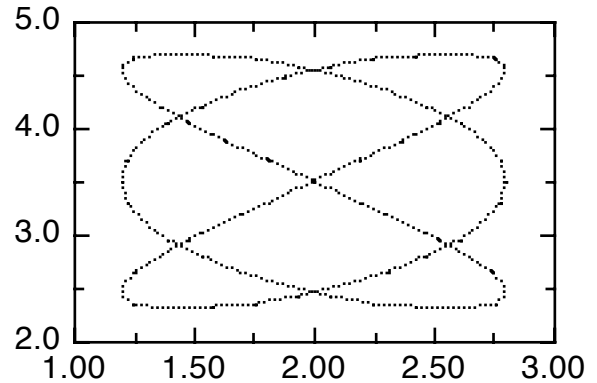

Creating and accessing graphs

`SetNewGraphRect`	sets the default size and position of graphs created with CreateNewGraph.
`CreateNewGraph`	creates a new graph in a drawing window.
`GetCurrentGraph,` `SetCurrentGraph, GetNextGraph`	obtain a unique identification number for a graph and use it to access different graphs.

Editing the current graph

`SetGraphAttributes`	set various options that determine the appearance of the current graph.
`SetLegendProperties`	set visibility, position and size of the legend of the current graph.
`GetGraphFrame, SetGraphFrame`	get/set the position and size of the current graph.
`GetGraphCoordinates`	returns the ranges of the main axes in the current graph
`SetRange, MakeTicks, SetLabelsFormat, SetAxisPosition, SetAxisAttributes`	change the range, ticks, position, labels format, and various drawing options for the current axis.
`MakeNewAxis, GetCurrentAxis, SetCurrentAxis, DeleteAxis`	create/kill coordinate axes in the current graph and change the current axis used to define a new curve or data set.
`ClearTicks, ClearLables, AddTick, SetLabel, SetLabelText`	define a custom list of tick marks and/or labels.

All the above calls access the current graph. To make a graph the current graph, double-click it while holding the command key down. From a program, you can use the call `SetCurrentGraph` to make a graph the current graph.

Some of the above routines use or change the axes of a graph. They access the *current x-axis* or the *current y-axis*. To make an axis the current x-axis, call `SetCurrentAxis(xAxis,i)`, where `i` is the number of the axis (`SetCurrentAxis(xAxis,2)` sets the current x-axis to X2). To make an axis the current y-axis, call `SetCurrentAxis(yAxis,i)`.

Calls that work on the current axes are `SetAxisPosition`, `SetLabelsFormat`, etc. The following code changes the position of the X2 axis of the current graph:

```
SetCurrentAxis(xAxis,2);    {2nd x-axis}
SetAxisPosition(xAxis,0.5);
```

Setting default parameters

`SetParameterProperties, GetParameterProperty`	Set and retrieve the value, name, limit and mode of a parameter

This routine is usually called in the procedure `Initialize` of a function. It allows to set the settings of a parameter that are given in the Parameter window.

Example: `SetParameterProperties` provides an alternative to the `inputs` statements (for external modules, it provides an alternative to setting the various default values and names by hand).

```
function foo;
  procedure Initialize;
  begin {initialization of param values, etc. }
      SetParameterProperties(param 1,
         value sin(pi/4), mode paramActive,
         name 'pi',min 0, max inf);
  end;
begin {function definition}
  y:=a[1]-sin(x);
end;
```

Using other functions or programs

`CallFunction,` `CallProgram`	call a function or execute a program.
`SetFunctionParam,` `GetFunctionParam,` `GetFunctionParamMode,` `GetFunctionParamName` `GetNumFunctionParams`	access other functions' parameters.
`SetFunctionProperites` `GetFunctionProperty` `SetProgramProperties,` `GetProgramProperty`	get and set function and program options, hide/show function in preview window
`GetFunctionName`	get name of current function
`SelectFunction,` `DeleteFunction, DeleteProgram`	select or delete a function or program
`GetGlobalData, SetGlobalData`	passing data between programs and/or functions
`LoadParameterSet,` `SaveParameterSet,` `UseParameterSet,` `DeleteParameterSet,` `AddParameterSet`	controlling the parameter set menu
`AddCommand`	add a command to the Prog menu
`AttachProgram`	attach a program to a drawing window

The following example program copies the active parameters of the current function to the first column of the current data window. It also calls a function called `ChangeUnit` to calculate new parameter values that it stores in the second column. Before using `ChangeUnit`, it sets the value of its first parameter to zero.

```
program CopyParams;
var i:integer;
    pa:real;
begin
SetFunctionParam('changeUnit',1,0.0);
for i:=1 to GetNumFunctionParams('') do
    if GetFunctionParamMode('',i)=active then
    begin
        pa:=GetFunctionParam('',i);
        data[i,1]:=pa;
        data[i,2]:=CallFunction('ChangeUnit',pa);
    end;
end;
```

Numerics on functions

`Integrate, TabulateIntegral, Derivative`	calculate the integral and the derivative of a function
`Roots, TabulateRoots`	calculate roots
`Fit`	set fitting, below.
`Optimize, Extrema, TabulateExtrema`	find extrema of a function by varying its x-value and/or its parameters
`Tabulate`	tabulate functions

Fitting

`Fit`	runs a fit.
`GetResult`	retrieves the results.

The following example runs a fit and prints some of the results:

```
program DoFit;
 var i, nrParams:integer;

begin
  Fit(function Sin, algorithm levenberg, xColumn 1,
    yColumn 2);
  Writeln('chi squared: ', GetResult(chiSquared));
  nrParams := GetResult(chiSquared);
  Writeln('number of parameters: ', nrParams);
  for i := 1 to nrParams do
    writeln('     ', GetResult(fittedParameter, i));
end;
```

Using Windows and Documents

NewDataWindow, NewFunctionWindow, NewDrawingWindow	open a new data, function, or drawing window
GetWindowID	obtain a unique identification number for a window from its title
FrontWindow, FrontmostWindow	obtain the ID of the document window in front of all others
GetWindowType,	check if a window is a drawing window, a data window, or a function window.
SetCurrentWindow, GetCurrentWindow, NextWindow	change the window currently used for program input/output.
GetWindowTitle,	access the title of a window.
SelectWindow	Bring window in front of all other windows
OpenFile	open a document and put it inside a new window
SaveWindow	save a window's contents into an existing or a new pro Fit document
GetFileDirectory	Get the directory where a given file resides.
SelectDirectory	Select a directory from a dialog box.
SetDefaultDirectory	Set the default directory used to save files without a full path name.
CloseWindow	close a window.
DataImportOptions, DataExportOptions	set the format for loading and exporting text files
SetWindowProperties, GetWindowProperty	set or retrieve the info-text, size, title, position, etc. of a window.
Compile, CompileText, DoScript	compile the definition of a function or program

PageSetup, Print   specify document format and print

Note: Windows are usually accessed by window ID. A window ID is a unique long integer number assigned to each window. You can obtain a window ID by calling `GetWindowID`, `FrontWindow`, `FrontmostWindow`, `GetCurrentWindow`. The following example sets the name of the frontmost data window to "favourite data":

```
program SetWindowName;
  var windowID:integer;
begin
  windowID := FrontmostWindow(dataType);
  SetWindowProperties(window windowID, name
    'favourite data');
end;
```

Note: The Results, Parameter and Preview windows always have the same window ID:

Results: window ID = –1
Parameters: window ID = –2
Preview: window ID = –4

The window IDs of data, text and drawing windows are always larger than 0.

String and character manipulation

Ord, Chr   convert between (real) ASCII codes and characters.

Length   returns the length of a string.

Delete   deletes parts of a string.

Pos   finds a pattern in a string.

InsertString   inserts a string into another at a selectable position

CopyString   copies a substring from a given string

UpperString, LowerString   converts between upper- and lower case strings.

NumberToString,   convert between numbers and strings.
StringToNumber

Tags

Tags are pieces of data that can be attached to a window, a program or pro Fit itself. A tag is identified by its name and its value can either be a string or a number. Tags are primarily used for passing data to or between programs/functions, for attaching custom data to windows or to pro Fit. Tags are identified by the object they belong to (a program, function, window, or pro Fit itself) as well as by their name.

GetTag, SetTag   get and set individual tags.

DeleteTag   deletes a tag.

Example:

```
SetTag(window 'myWindow', tag 'tag 1', value 13); {saves the tag}
GetTag(window 'myWindow', tag 'tag 1', value x);  {reads the tag to x}
```

From Apple script, you can use

```
get value of tag "tag 1" of window "myWindow"
```

Getting and Setting "Properties" of various pro Fit objects

pro Fit 5.5 supports a general mechanism to retrieve or set various properties of various objects, such as the coordinates of shapes in a drawing window, the title or size of a window, etc. The following routines provide access to the properties of drawing shapes, windows, function and programs, and pro Fit itself.

`SetShapeProperties,` `GetShapeProperty`	generic routines for checking, and changing any type of shape in a drawing window. (Every object in a drawing window, e.g. rectangles, groups, graphs, legends, etc, is a shape.)
`SetWindowProperties,` `GetWindowProperty`	set or retrieve the info-text, size, title, position, etc. of a window.
`SetDataWindowProperties,` `GetDataWindowProperty`	get and set the number of columns and rows in the current drawing window.
`SetFunctionProperties` `GetFunctionProperty` `SetProgramProperties,` `GetProgramProperty`	get and set function and program options, hide/show function in preview window
`SetOptions` `GetOption`	set and retrieve various options of pro Fit.
`SetParameterProperties` `GetParameterProperty`	set and retrieve the value, name, limit and mode of a parameter.

Miscellaneous auxiliary routines

`Random`	returns a random number between 0 and 1.
`Invalid`	checks if the result of a calculation is a valid number.
`TickCount`	return the number of ticks (1/60 seconds) since start-up.
`GetDateTime,` `DateString, TimeString`	return today's date and time, either as a number of seconds since 1.1.1904 or as strings.
`NumToDateTimeStr,` `DateTimeStrToNum`	convert data & time numbers (seconds since 1.1.1904) into data & time strings and vice versa.
`NumToRelTimeStr,` `RelTimeStrToNum`	convert relative times (seconds) into relative time strings and vice versa. A relative time can be the difference of two dates.
`Beep`	lets your computer emit an alert sound.
`SpeakString`	lets your computer speak out loud a text string.
`Button, KeyPressed`	check the mouse button and the keyboard
`GetClickedCoord`	find the last clicked point in the current drawing window.
`MarkedX, MarkedY,` `GetMarkedCoord`	find the position of coordinate markers in the preview window
`Undo, Cut, Copy, Paste,` `Clear, SelectAll`	execute edit menus
`DoMenu`	execute a menu command
`Capture`	redirect output of results window to file
`SetWaitTitle, SetWaitText`	set the text displayed in pro FIt's progress window, shown during lengthy operations.
`SetOptions, GetOption`	set and retrieve various options of pro Fit.

The compiler

When adding a definition to the list of functions or programs of pro Fit, the definition text is translated into machine code that can be executed by your computer. This results in a very fast execution speed of programs and functions.

```
  functions              programs

  y := a[1]*sin(x)       for i:=1 to 10 do
                           data[i,1] := 0;


                  compiler


                    code

                  00FA 2CC3
                  3008 299S
                  8001 FF29
```
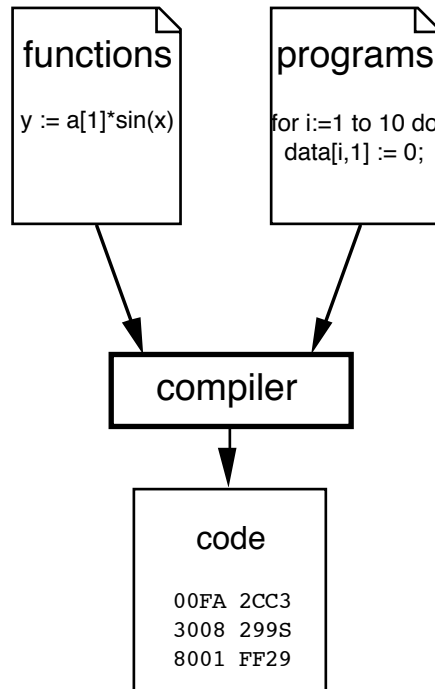
The translation of your definitions into machine code is carried out when you choose Add to Menu from the Prog menu or if you click the button "Add" in the toolbox of the function window.

Any changes that you make to your definition *after* compilation will not affect the function or program as it was added to pro Fit's menus. To update your changes, you must choose Add to Menu again.

Comparison to standard Pascal

The programming language used to define functions and programs in pro Fit is closely related to the Pascal programming language. However, to keep it simple and to allow the generation of fast code, some restrictions are present. However, there are also some extensions with respect to standard Pasca. The most important differences to standard Pascal are:

- You cannot define your own **data types**.
- All numeric types (except complex) are interpreted as floating point numbers. Boolean expressions are evaluated as floating point numbers (a 0.0 representing `false`, any non-zero value representing `true`). No **records**, **structures**, or **pointers** are supported.
- **Arrays** are one dimensional.
- **Case** statements are not supported.
- Nested declarations of functions or procedures are not supported.
- Optional parameter predefined procedures and functions are supported.
- A general purpose complex type is supported.
- General purpose matrix and vector types are supported.

External functions and programs

Even though pro Fit's definition language is very powerful, it does not offer the full versatility of a special purpose programming language. It only supports one dimensional arrays(except `data[i,j]`), records, pointers, etc. In addition, it does not support access to the Macintosh toolbox routines. If you do need any of these features or if you want to write a large program or function for pro Fit where execution speed is crucial, you should write your definition in any compiler of your choice and add the generated code to pro Fit. This process is called 'writing a plug-in'. See Chapter 10, "Working with plug-ins" for details.

Debugging Window

pro Fit provides a powerful debugging environment for the development of your programs and functions. For using this environment, check the option "Debug" at the top of its window. When you run the program or function, its debug window will show up:
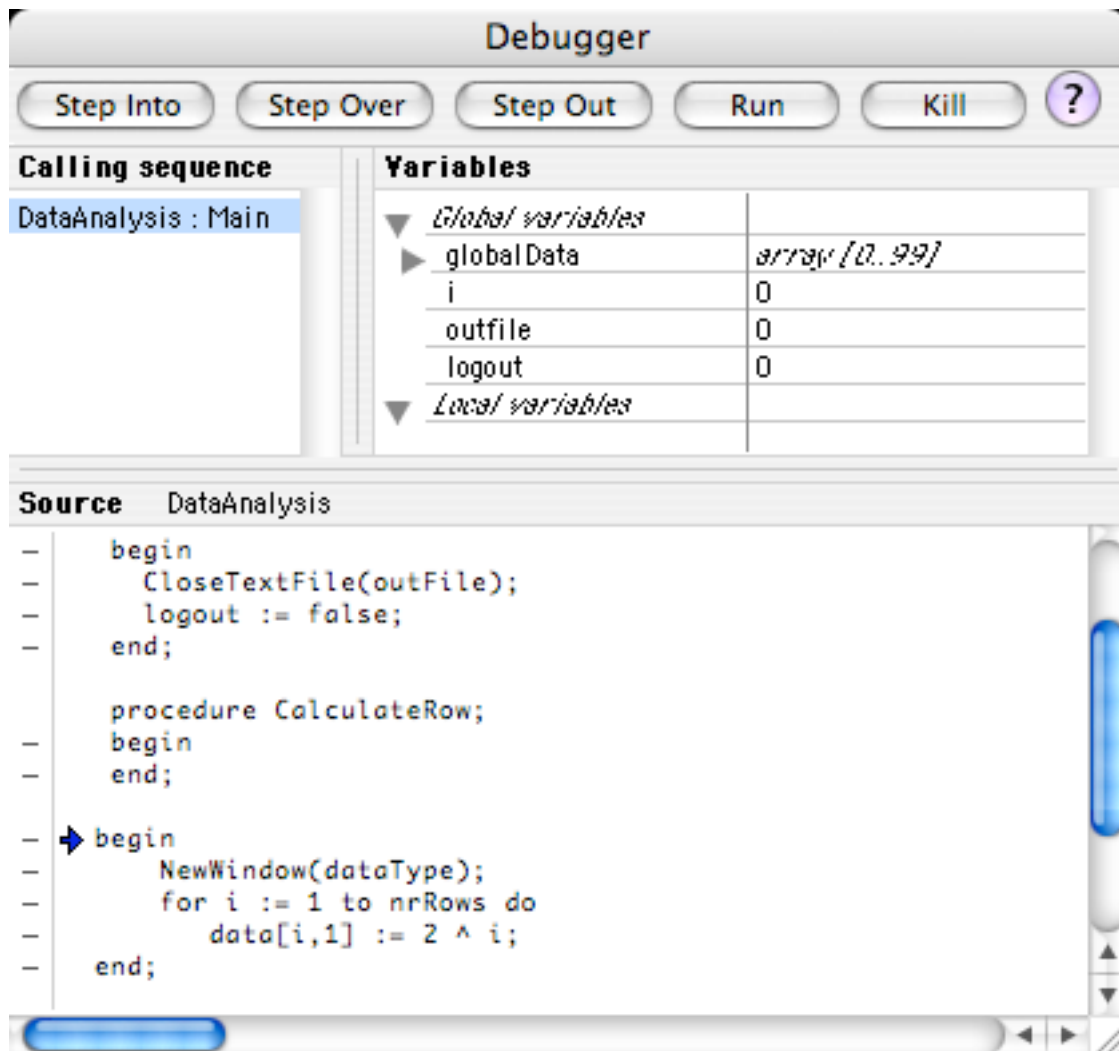
Now, you can step through your program, view and modify its variables, set breakpoints, etc.

Initially the program stops at the first line of code that is executed. (Note: Some parts of your program may already be called right after compilation, such as the procedure `Initialize`. In this case, the debugging window will come up right after compilation to let you debug these parts of your code.)

The debug window has four parts:

- At the very top, there's a button bar. The significance of each button is explained below.
- At the top left, the "Calling sequence" is shown. It shows through what chain of procedures and functions pro Fit went in order to reach this particular point in your code. Note that you may step through more than one of your programs and/or functions, in the case they call each other.
- At the top right, the variables that are valid at this point are displayed. You can watch and modify their values. Just double-click a value to change it. Clicking onto the small triangles lets you view the elements of arrays and matrices.
- At the bottom, the source of the program or function is displayed, with an arrow showing the current location.

The buttons at the very top let you control operation of the prosecution of your code:

- Click **Step Into** or **Step Over** for advancing one step in your code. When you click Step Into and the next step is a local function or a procedure, pro Fit steps into this procedure and stops at the first instruction there. If you click Step Over and the next step is a function or procedure, pro Fit will execute it and stop again right after. If the next step is not a function or procedure, Step Into and Step Over just advance by one step.
- Click **Step Out** if you are in the midst of a local function or procedure and you want pro Fit to stop when execution returns from this function or procedure, i.e. you do not want to stop again until the function or procedure is terminated.
- Click **Run** to continue operation to the next breakpoint or (if there is no more breakpoint) to the end of your code.
- Click **Kill** to abort execution of your function or program.

You can set "breakpoints" by clicking into the left margin of the source code in the debug window. Red dots mark the breakpoints. To remove a breakpoint, click it again. When you run a program or function and pro Fit encounters such a breakpoint, execution is interrupted and the debug window comes

Using pro Fit plug-ins

After you have added a function or program to the menus, you can save its compiled code as a separate file for later use. This file is called a plug-in.

You can also create plug-ins in an external compiler. pro Fit comes with a set of plug-ins for different tasks. You can use them to add functionality to your copy of pro Fit according to your needs. See Chapter 10, "Working with plug-ins" for an explanation on how to build plug-ins.

This section explains how to use such plug-ins.

Saving functions and programs

To save a function or program as a plug-in, choose **Save as Plug-In** from the Customize menu to see a submenu with all the functions and programs that can be saved as plug-ins.
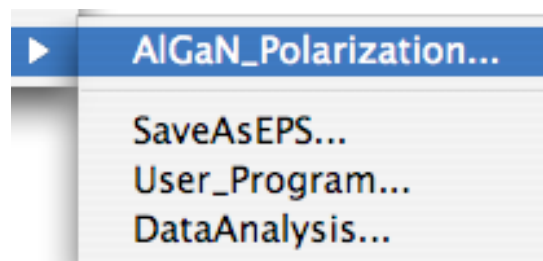
This sub-menu has two sections divided by a horizontal line. The first section lists the functions, the second section the programs. Choose the function or program you want to save as a plug-in, and pro Fit will ask you where you want to save it. Note that you can only save functions and programs that you compiled in pro Fit – you cannot save built-in functions or plug-ins.

The resulting file is a pro Fit document. You can load it by using the Load Plug-in command or by double clicking it from the Finder.

Loading Plug-ins

Choose "Load Plug-in..." from the Customize menu to load a plug-in. You are asked to locate the plug-in.

The command "Load Plug-in..." can also be used to load compiled Apple Scripts. See Chapter 11, "Apple Script" for details.

Removing functions and programs from the menus

To remove a function or a program (or an Apple Script) from pro Fit's menus, choose "Remove from Menu" from the Customize menu. A submenu lists all the functions and programs that can be removed from the menus. Select the name of the function or of the program you want to remove.

Note: you cannot remove any of pro Fit's built-in functions (Spline, Polynom, etc.).

Loading plug-ins automatically on startup

Imagine you have one or more plug-ins or Apple Scripts that you use often. You can make them available automatically whenever you start pro Fit.

Put the plug-ins you want to add permanently to pro Fit into a folder named "pro Fit plug-ins". This folder must be located in the same folder as pro Fit's or in the Preferences folder of your System Folder. (When you create the folder "pro Fit plug-ins", type the name exactly as given here, otherwise pro Fit will not find it.)

Whenever pro Fit starts up, it checks if a folder named "pro Fit plug-ins" is located in the same folder as the application itself and tries to load all plug-ins it finds there. Then pro Fit looks for a folder "pro Fit plug-ins" in the Preferences folder of the Library folder and again tries to load all plug-ins it finds there.

If you are running pro Fit directly from a server, the modules found in the "pro Fit Plug-ins" folder in the application folder on the server will be available to all users, the plug-ins in the "pro Fit plug-ins" folder of your system's Preferences folder will only be available to you.

You can also add plug-ins to the pro Fit application directly. To do so, select the pro Fit application in the Finder and choose Get Info from the File menu. In the window that appears, go to the "Plug-ins" section and click the button "Add" to select a plug-in to add.

Note: You can also place drawing, data or function files into the "pro Fit plug-ins" folders. The names of these files will automatically appear in the Prog menu. Choosing the name from that menu will open the file.

Loading a set of plug-ins together with a new preferences file

In multi-user environments different users might want to use the multi-preferences-file mechanism provided by pro Fit.

The pro Fit preferences file holds the default settings and other information for many pro Fit's options. Different users may want to use different preferences files. pro Fit normally uses the preferences file found in the Preferences folder inside your Library folder. It is possible, however, to start pro Fit by double clicking another preferences file, or to switch to a new preferences file while pro Fit is in use by choosing **Preferences...** from the File menu. This allows each user to use his own set of preferences. See Chapter 13, "Preferences" to learn how to use preferences files.

pro Fit provides a mechanism that allows users to load their favorite plug-ins together with their preferences file: whenever a preferences file is opened, pro Fit looks for a folder named "pro Fit plug-ins" in the same folder as the preferences file and loads all the plug-ins it contains.

To take advantage of this mechanism, simply put your preferences file and pro Fit plug-ins folder inside a common folder. Whenever pro Fit opens the preferences file, it also loads all the plug-ins found in the "pro Fit plug-ins" folder.

Attaching programs

Programs can be attached to drawing windows. Such programs are called whenever there is a user-interaction with drawing windows, *e.g.* when they are clicked, opened, closed, etc. This feature is useful when using the drawing window to design an interface for a program. The attached program can then read the actions of a user, and interpret them.

To attach a program to a drawing window or to modify an attached program, bring the drawing window to front, choose GetInfo from the File menu and check "Show program window". Then click OK. Alternatively click into the drawing window while holding down the control key and choose "Show program window" from the contextual menu. A window with the source of the attached program appears.

Once you have defined the program, choose "Compile" from the Customize menu. The program is compiled and its code is attached to the window.

A program attached to a window (an "attached program") communicates with pro Fit using **tags** (see below). An attached program should always check its tag `msgWhy` to find out why it was called. If this tag contains an unknown stringValue, the program should do nothing. Otherwise, it should take some action according to its needs.

The following code-snippet retrieves the "`msgWhy`" tag:

```
    var msgWhy:String;
    ...
    GetTag(program '', tag 'msgWhy', stringValue msgWhy);
```

The tag `msgWhy` can currently have the following stringValues:

`'clicked':`	The drawing window was clicked. In this case the tag "`msgShape`" will have a `stringValue` set to the name of the clicked shape (if a shape was clicked) or will have an empty stringValue if no shape was clicked. The tags `'msgClickedX'` and `'msgClickedY'` contain the clicked coordinates.
`'control clicked':`	A control shape was clicked successfully. In this case, the tag 'msgShape' has a stringValue set to the name of the clicked shape, The tags `'msgClickedX'` and `'msgClickedY'` contain the clicked coordinates.
`'control keydown start':`	A control receives keyboard input. This tag message is sent before the key is processed. In this case, the tag `'msgShape'` has a stringValue set to the name of the shape, The tag `'msgCharCode'` has a stringValue of length 1 giving the char code of the pressed key. (For easier comparison there are the following charCodes constants predefined: `charHome, charEnter, charEnd, charBackspace, charTab, charLf, charPageUp, charPageDown, charCr, charEsc, charArrowLeft, charArrowRight, charArrowUp, charArrowDown, charDelete`. The tag `'msgKeyCode'` has a stringValue of length 1 giving the key code of the pressed key. The tag `'msgModifiers'` has a value set to the keyboard modifiers (it tells, *e.g.* if the option-key was pressed). You can change the `msgCharCode, msgKeyCode` and `msgModifiers` tag to change the keyboard event before it is processed. (For easier comparison there are the following modifier codes predefined: `modButtonState, modCommand, modShift, modAlphaLock, modOption, modControl`.) You can set `msgCharCode` to an empty string to suppress the event.
`'control keydown end':`	A control has received keyboard input. Called after the key is processed. Same parameter as for message "`control keydown start`".
`'opened':`	The drawing window was opened.
`'save':`	The drawing window will be saved.
`'close':`	The drawing window will be closed.
`'command':`	A command added by the procedure `AddCommand` has been called. The tag "`msgCommand`" contains the name of the command.

`'idle':`	The program is being called because the value in its property `'idleCallTime'` corresponds to the present value of TickCount.

In addition to the tag "`msgWhy`", attached programs can always rely on the presence of the `'msgOwnerWindow':` The value of this tag is the ID of the window to which the program is attached.

An attached program should therefore look like this:

```
program attached;
  var msgWhy: String;
begin
  GetTag(program '', tag 'msgWhy', stringValue msgWhy);
  if msgWhy = ... then        check here for known tags
  ...
end;
```

It is also possible to attach a program from another program using the call `AttachProgram`.

Working with control shapes

As explained in Chapter 7, drawing windows can contain "control shapes", such as buttons or checkboxes. The following is a list of all control shapes and of the most important properties they have. These properties can be read by calling `GetShapeProperty` and modified through `SetShapeProperties`.
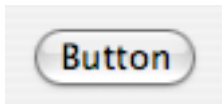
Buttons: These are simple objects that hilite when clicked. Properties:

  `active`: Set to true if the button can be clicked. Set to false if it is grayed and cannot be clicked.

  `value`: Usually 0. Set to 1 for hiliting the button.

  `text`: The text that appears in the button.
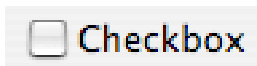
Checkboxes: They automatically change their state when they are clicked. Properties:

  `active`: Set to true if the checkbox can be clicked. Set to false if it is grayed and cannot be clicked.

  `value`: 0 if not checked, 1 if checked.

  `text`: The text that appears beside the checkbox.
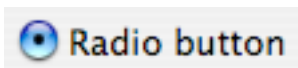
Radio buttons: They are checked when they are clicked. They usually come in groups. The program that manages the radio buttons is responsible for unchecking all other radio buttons when one radio button is clicked. Properties:

  `active`: Set to true if the checkbox can be clicked. Set to false if it is grayed and cannot be clicked.

  `value`: 0 if not checked, 1 if checked.

  `text`: The text that appears beside the radio button.

Text fields: These are shapes that contain editable text. Generally, text fields can be edited.

  `active`: Set to true if the field can be edited. Set to false if it cannot be edited.

  `value`: The numeric equivalent of the text appearing in the field. Use the function `Invalid` to check if the text corresponds to a value number.

  `text`: The text that appears in the edit field.
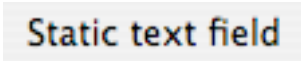
Static text fields: These are shapes that contain non-editable text. Properties: same as for text fields, except that active has no influence on the shape's editability.
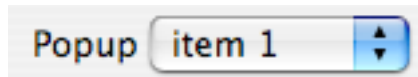
Popup menus: Popup menu shape have several "values" which can be selected by choosing them from a pop-up menu. Properties:

  `active`: Set to true if the pop-up can be clicked. Set to false if it cannot be clicked and is grayed.

  `value`: The currently selected item in the pop-up menu. 1 is the first item, 2 the second item, etc.

  `text`: The text that appears to the left of the pop-up.

  `menuItems`: The menu items, separated by semicolons.
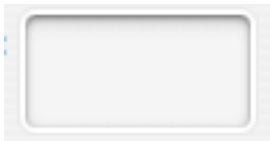
Wells: These shapes are usually used as background for other objects, e.g. a graph. They consist of a white rectangle.

For further properties that you can use for controlling these shapes, see the description of `GetShapeProperty` and `SetShapeProperties` in pro Fit's on-line help.

To use control shapes, you first must draw them in a drawing window. Then you write a program that manages them and attach it to the window. Finally, you must switch the window to "dialog mode". The following is a simple example that shows this procedure.

1. Open a new drawing window and create a button named "Multiply"

To do this, choose the button tool from the windows toolbox. Then click into the drawing window. A dialog box appears where you can define the text that appears on the button. You can also define a name for the button, that we will later use for accessing the button from a program. In this example, set the button text to "Multiply by 2" and its name to "Button".
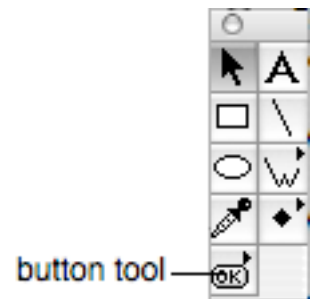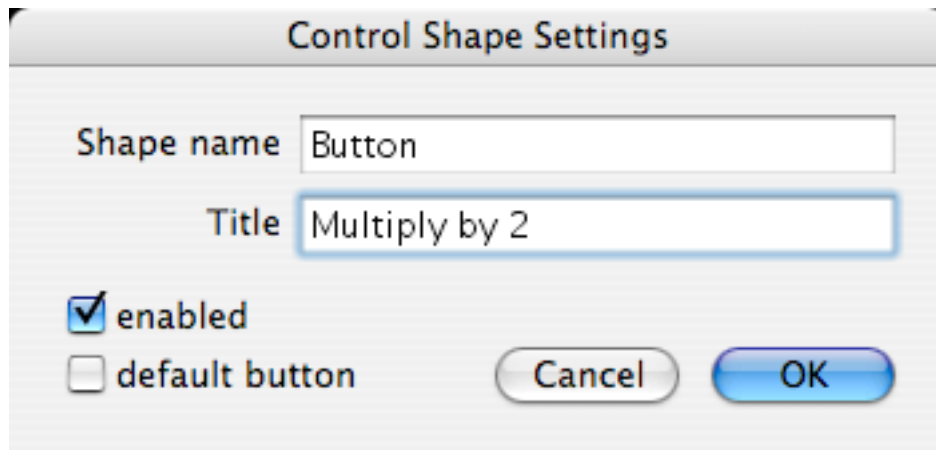
Click OK, and the button will appear in the drawing window.

2. Create an edit field named "Number"

Now, click the button tool again and hold the mouse down until a poup menu appears. Choose "Text Field". Now, click into the drawing window and enter "Number" for the shape's name. Then click OK.

You now should have a drawing window with an edit field and a button. Arrange these items as you wish, then save the file.

The window might e.g. look like this:
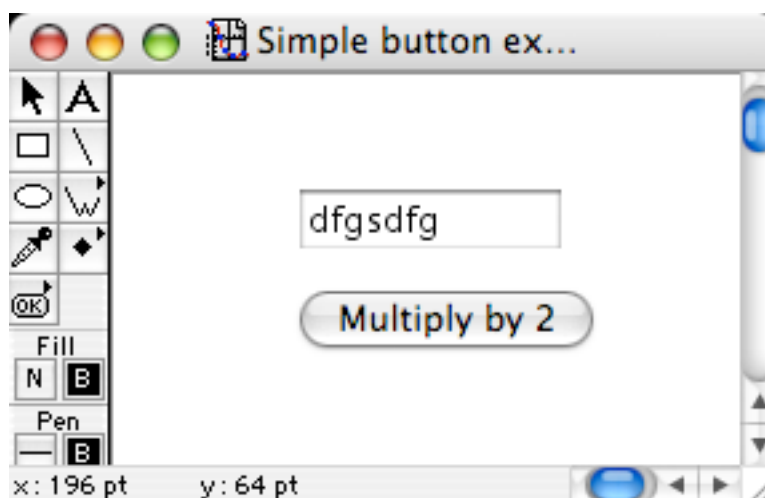
3. Switch the window to dialog mode

To do this, hold down the control key while clicking anywhere into the window and choose "Display As Dialog". Alternatively, choose "Get Info..." from the

File menu and check the option "Display As Dialog".
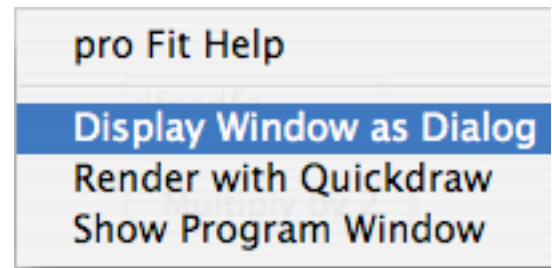
The window now looks like a dialog box.
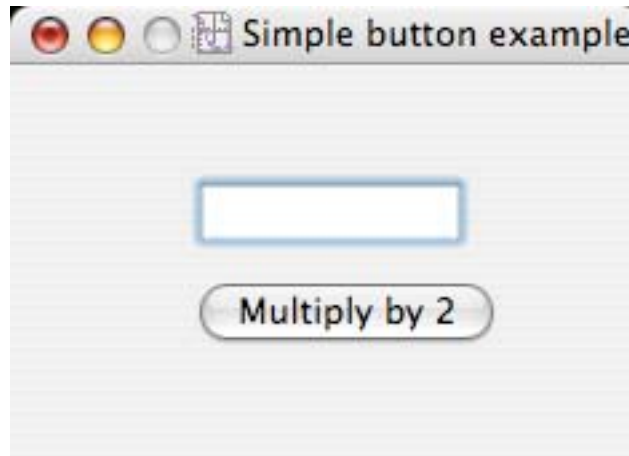
4. Attach the program

To attach the program, again hold down the control key while clicking anywhere into the window and choose "Show Program Window". Then, enter the following program:

```
program attached;
    var msgWhy: String;
        msgShape: String;
        x: real;
begin
  GetTag(program '', tag 'msgWhy', stringValue msgWhy);
  if msgWhy = 'control clicked' then
  begin
    GetTag(program '', tag 'msgShape', stringValue msgShape);
    if msgShape = 'Button' then
    begin
      x := GetShapeProperty('Number', value);
      if not Invalid(x) then {if valid number}
        SetShapeProperties(shape 'Number', value x*2);
    end;
  end;
end;
```

Hit Command-L to add the program to the window.

Now, your "dialog box" is ready to use. Enter a number in the edit field, then hit "Multiply by 2".

Notes:
- If you want to modify the items in the dialog window, switch it back into drawing mode. To do this, hold down the control key and choose "Display As Drawing". For changing the text of a shape or its name, double-click it. Alternatively, select it and choose "Shape Settings..." from the Draw menu.
- As a shortcut, you can change some properties of the items when the window is still in dialog mode. To do so, hold down the command key and double-click the item you want to modify.

10 Working with plug-ins

This chapter explains how to add *plug-ins* to pro Fit. Plug-ins are files containing the computer code for a pro Fit function (to appear in the Func menu) or a pro Fit program (to appear in the Prog menu).

pro Fit comes with a number of ready-to-run plug-ins containing useful functions or programs. The next section tells you how you add them to pro Fit.

See the sections "Creating a plug-in" and "Writing a plug-in" for a detailed explanation of how to create your own plug-in.

Loading a plug-in

To add a plug-in to pro Fit:

1. Select Load Plug-in from the Customize menu.

You are asked to locate your plug-in:

2. Choose the plug-in you want to load and click "Open".

pro Fit checks if a plug-in can be found in the file you have selected. If yes, it is loaded. If the plug-in is a function, it is added to the Func menu. If it is a program, it is added to the Prog menu.

Instead of loading a plug-in by choosing Load Plug-in, you can double-click its file. (For this, the 'file type' and 'creator' of the file must be 'ftCD' and 'NLft', or it should have one of the following extensions: code, .fitcode .plugin, .fitplugin .proFitCode, or .proFitPlugin).

> If you have loaded a plug-in and you subsequently change it (e.g. by recompiling it) you must **remove the loaded plug-in from pro Fit before loading its new version**.

To load your plug-ins automatically at start-up, put them into a folder called "pro Fit plug-ins" located in the same folder as the application itself or in the Preferences folder of the Library folder. See the end of Chapter 9, "Defining functions and programs", for a more detailed discussion of how to work with pro Fit plug-ins.

The rest of this chapter explains how you can write plug-ins using your own compiler.

Creating a plug-in with a compiler

You need the following to write a plug-in:
• Some experience in programming.
• A compiler (such as Metrowerks® C/C++ or Aplle's XCode).

To create plug-in, proceed as follows:

1. Locate the development files

You will find all files that you need to program a plug-in in the folder Example Files/ Programming 7 - plug-ins/ of the pro Fit distribution package.
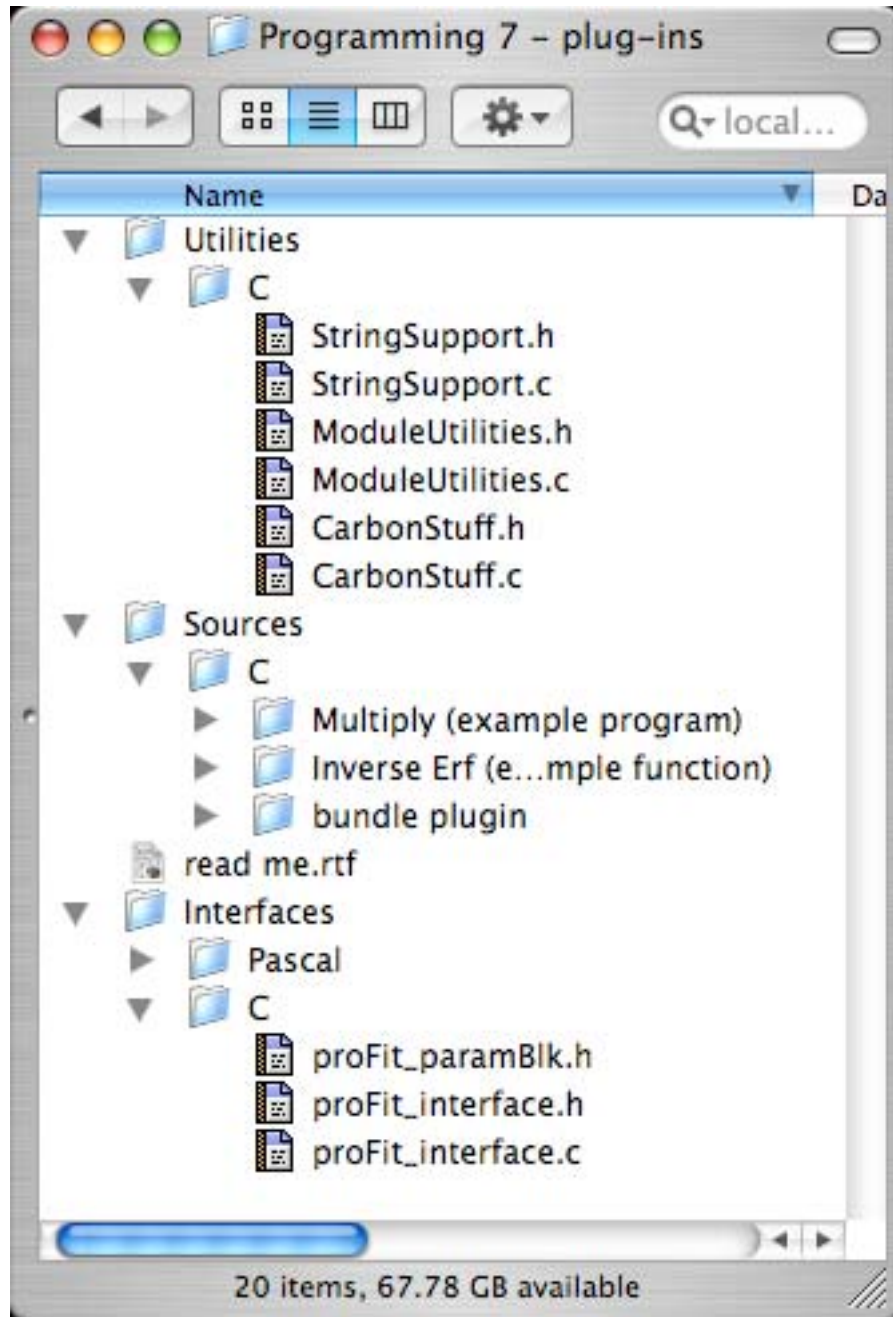
The folder Interfaces contains files that allow your plug-in to communicate with pro Fit.

The folder Sources contains example projects. Multiply and Inverse Erf are a simple program and a simple function plug-in for Metrowerks Codewarrior. Bundle plugin is an example for a simple program plug-in that can be compiled under XCode as well as Codewarrior.

2. Copy one of the example projects

If you are programming with XCode, make a copy of the "bundle plugin" folder. If you are using Codewarrior, you can copy any of the three example projects, whichever comes closest to what you do. Note that the copy should reside in the same folder as the example projects.

3. Build your code

Modify the code in the duplicated folder according to your needs.

3. Build the plug-in

The plug-in should be created in a file having the type "ftCD" and the creator "NLft". If you are building the file under MacOS X on a non-HFS volume (which is not recommended), add the extension ".fitcode" or ".proFitCode" to the end of the file's name, e.g. "myPlugin.fitcode". The example projects are set to generate the correct file types.

4. Link the plug-in to pro Fit

To do this, either double-click the file you have built or load it from pro Fit by choosing Load Plug-in... from the Customize menu.

The following gives some hints for creating plug-ins with some of the most common compilers. Note that there are sample "project" and "make" included with the pro Fit package.

Writing an a plug-in with an external compiler

Routines to be modified

The following table lists the routines defined in ProgramTemplate.c/p and FunctionTemplate.c/p that can or should be modified by the user. Functions or procedures that are only used by advanced programmers are marked with a [†]:

function name	modify if defining a
SetUp	program or function
CleanUp [†]	program or function
InitializeProg [†]	program
Run	program
InitializeFunc [†]	function
Func	function
Derivatives	function
First [†]	function
Check [†]	function
Last [†]	function

In the following section, we first describe the routines `SetUp` and `CleanUp` that are used for both types of plug-ins. Then we discuss the routines only used in external programs, then the routines only used in external functions.

All the following routines have a parameter called `pb`. It is a pointer to a struct of type `ExtModulesParamBlock`. Most users won't need the information stored in it. Advanced

programmers can refer to the section "Global variables" for more information about data to be accessed through `pb`.

Routines to be defined in functions and programs

```
void SetUp (short* const moduleKind, Str255 name,
    long* const requiredGlobals, ExtModulesParamBlock* pb);
```

This routine is called when your plug-in is linked to pro Fit. It must return the following values:

- `moduleKind` must be set to the constant `isProgram` if your plug-in is an external program, and to `isFunction` if your plug-in is an external function.

- `name` must be set to the name of your plug-in. If you are programming in C, you must make sure that you return a Pascal string. For this purpose, you can use the function `SetPascalStr` that is defined in proFit_interface.c:
  ```
  SetPascalStr(name,"\pmyName",255);
  ```
  (The last parameter is the maximum length of the resulting string.)

- `requiredGlobals` should usually be set to 0. Advanced programmers can set it to the size (in bytes) of a global data buffer they want to have allocated. If `requiredGlobals` is returned with a value > 0, pro Fit allocates a block with the corresponding number of bytes and stores a pointer to it in `pb->globals`. `pb` is a pointer to a record called `ExtModulesParamBlock` and is passed to all routines called by pro Fit.

  Note that memory allocated in this way is deallocated automatically when your plug-in is unlinked from pro Fit – you must not deallocate this memory yourself.

  Generally, you should use static variables for global memory instead of using pb->globals.

```
void CleanUp (ExtModulesParamBlock* pb)
```

CleanUp is called when pro Fit is quitting or when your plug-in is removed from pro Fit. In most cases, you won't have to do anything here. Advanced programmers may wish to deallocate some special memory, to close a port or to clean up other stuff here.

Routines to be modified in external programs only

```
void InitializeProg (ExtModulesParamBlock* pb)
```

This routine is called before a program is run for the first time. Most users can leave it empty. Advanced programmers may wish to allocate some memory, open a port, initialize global (static) variables, etc. here.

```
void Run(ExtModulesParamBlock* pb)
```

This routine is called when your program is executed. It should hold your program's main code.

Routines to be modified in external functions only

> ! An important note about parameter indices: When accessing arrays that hold values, names, etc. of the parameters, such as `a[], a0.names, mode[], dyda[]`, the index `i` ranges from 0 to 127 in C

```
void InitializeFunc (Boolean* const hasDerivatives,
    Str255 descr1stLine, Str255 descr2ndLine,
    short* const numberOfParams,
    DefaultParamInfo* const a0, ExtModulesParamBlock* pb)
```

This routine is called once after your external function has been linked to proFit. It must return some default values and information about the function. Advanced programmers may also use it for initialization of global (static) variables, memory allocation, etc.

`InitializeFunc` should return the following data in its parameters:

- `hasDerivates` must be set to `true` if you want to calculate some derivatives of your function with respect to its parameters yourself (in the function `Derivatives` described below). Any derivative you don't calculate will have to be calculated numerically by proFit. If you set `hasDerivates` to `false`, all derivatives will be calculated numerically and the function `Derivatives` will be ignored. (The derivatives are used for nonlinear fitting.)
- `descr1stLine`, `descr2ndLine`: These two strings are displayed in the parameters window and should give a short description of your function. (C programmers should use the function `SetPascalStr` described under `SetUp`, above, for setting these strings.)

- `numberOfParams`: Here you should return the number of parameters of your function (up to 128).

- a0: This is a pointer to a structure that defines the default values, modes, names and limits of your parameters. You can leave this record unchanged if you want to use the default values.
  The following table lists the values that can be set in a0 for each parameter i:

Pascal notation [1]	C notation [2]	contains
`a0.value^[i]`	`(*a0->value)[i]`	Default value
`a0.mode^[i]`	`(*a0->mode)[i]`	Default mode, set to `active` (varied during fitting), `inactive` (not varied during fitting), or `constant` (cannot be fitted)
`a0.name^[i]`	`(*a0->name)[i]`	Parameter name, a Pascal string of length `maxParamLength`. [3]
`a0.lowest^[i]`	`(*a0->lowest)[i]`	The lower limit for a parameter. By default, this value is -INF.
`a0.highest^[i]`	`(*a0->highest)[i]`	The upper limit for a parameter. By default, this value is INF.

[1] In Pascal, indices for these arrays run from 1 to 128
[2] In C, indices for these arrays run from 0 to 127
[3] C programmers should set the name by calling the function `SetPascalStr` with a maximum string length of `maxParamLength`. Example:
```
SetPascalStr((*a0->name)[0],"\pname",  maxParamNameLength);
```

`void Func (double x, ParamArray a,double* const y,`
  `ExtModulesParamBlock* pb)`

This procedure is called to calculate the return value of your function. It has the following parameters:
- `x`: The function's independent variable.
- `a`: The function's parameters `a[i]`. Note that the index `i` ranges from 1 in Pascal but from 0 in C.
- `y`: The function's return value to be calculated from `x` and `a`.

`void Derivatives(double x, ParamArray a, ParamArray dyda,`
  `ExtModulesParamBlock* pb)`

This routine calculates the partial derivatives of your function with respect to its parameters. You can leave this routine empty if you don't need it, or you can calculate only some derivatives. You don't need to calculate all of them. pro Fit will check if you did not calculate a derivative and will calculate it numerically. Set `hasDerivatives` to false in `InitializeFunc` if you are sure that you will never want to calculate any derivatives yourself. (Note that a call of `Derivatives` with a given x-value is always preceded by a call of `Func` with the same x-value – therefore, you might save a temporary result in `Func` for later use in `Derivatives`. See also Chapter 9, "Defining functions and Programs".)

Parameters:
- `x`: The function's independent variable.
- `a`: The function's parameters `a[i]`. Note that the index `i` ranges from 1 in Pascal but from 0 in C.
- `dyda[i]`: The partial derivatives to be returned.

`void First (ParamArray a, ExtModulesParamBlock* pb)`

This routine is called whenever the parameters a have changed *before* `Func` is called. In most cases, you can leave it empty. Advanced programmers can use `First` for speeding up your function by

evaluating temporary results that only depend on your function's parameters but not on its x-value (for more information: see the description of First in Chapter 9, "Defining functions and Programs").

Parameters:
- a: The function's parameters `a[i]`. Note that the index `i` ranges from 1 in Pascal but from 0 in C.

```
short Check(short paramNo, DefaultParamInfo* const a0,
    ExtModulesParamBlock* pb)
```

Check is called whenever the user has entered a value in the Parameters window. In most cases, you can leave Check empty, returning the value `good`. Advanced programmers can use it for improving the parameters window's user interface. Applications of Check are described in Chapter 9, "Defining functions and Programs").

Parameters:
- paramNo: This is the index of the parameter that the user has changed (1..64 in Pascal, 0..63 in C).
- a0: This is a record (in C: a pointer to a struct) that defines the default values, modes, names and limits of your parameters as they appear in the parameters window. The values that you can access or change in this data structure are listed under the routine `InitializeFunc` above.

Check should return one of the following values:
- good if the new parameter is to be accepted
- update if the new parameter is to be accepted but the parameters window must be redrawn (because Check changed some values in a0)
- bad if the new parameter cannot be accepted.

```
void Last (ExtModulesParamBlock* pb)
```

This routine is called whenever an operation that has used your function (such as a command for fitting) is done. In most cases, you can leave this procedure empty. Applications of Last are given in Chapter 9, "Defining functions and Programs".

Predefined constants and types When writing a plug-in, you can (and must) use several predefined constants, types and procedures (or functions). They are defined in profit_interface.h and proFit_paramBlk.h. This section describes some of the most important things defined in these files.

> The definitions in these files should not be changed. Doing so might cause incompatibilities with the present or future versions of proFit.

General remarks:
- *Strings* passed between proFit and a plug-in are always Pascal strings (and not C strings). If you are programming in Pascal, you won't have any problems with this. If you are programming in C, you must remember that a Pascal string must be introduced by "\p" (example: "\pMyString"). For assignments, you can use the function `SetPascalString` described earlier in this chapter.
- *Records* (*structs*) passed between proFit and a plug-in always use "68k-alignment". Therefore, for compatibility with Power Macintosh compilers, definitions for C structs are always preceded by

```
#if defined(powerc) || defined (__powerc)
#pragma options align=mac68k
#endif
```
and followed by
```
#if defined(powerc) || defined (__powerc)
#pragma options align=reset
#endif
```
- *Parameter indices* under Pascal always run from 1 to `maxNrParams`, in C they run from 0 to `maxNrParams—1`.

The following lists the most important constants and types.

Global variables

Global variables (or static variables, as they are often called by C programmers) are variables that remain statically in memory. Their values are preserved between individual calls to your plug-in.

In XCode and CodeWarrior you can define global variables in the way you are used to: In Pascal, you declare them globally within your unit – in C, you declare them outside your functions or, if you declare them inside a function, you declare them as `static`.

Procedures provided by pro Fit

pro Fit offers a list of functions and procedures that can be called by your plug-ins. If you are programming in Pascal, they are defined in the interface of the file proFit_interface.p. If you are programming in C, they are defined in the header file pro Fit_interface.h. Their implementation can be found in the files pro Fit_interface.p or proFit_interface.c, respectively.

Most of the functions and procedures provided by pro Fit for plug-ins are the 1:1 equivalents of the ones that can be used when defining a function or program with pro Fit's definition language. Refer to pro Fit's on-line help for more information on the individual routines.

11   Apple Script

Introduction

Apple Script is a language for scripting applications on the Macintosh. It provides a common technique for automating tasks, exchanging data, and process remote control.

You can use Apple Script with pro Fit. Note, however, that pro Fit cannot *create* (i.e. compile) an Apple Script. To use Apple Script with pro Fit, you need an Apple Script compiler, such as Apple's Script Editor (you can find it in the folder Apple Script of your Application's folder). You enter the script in the script editor and compile it there.

Once the script is compiled, you can either run it from your script editor, or you can save it in its compiled form. (When using Apple's Script Editor, choose "Save As..." from the "File" menu, choose the type "Compiled script" and save the script.) Such a compiled script can be loaded into pro Fit: Choose "Load Plug-in..." from the Customize menu and select the compiled script. It is added to the Prog menu.

compiled script

In the following, we give some examples for scripting pro Fit through Apple Script. Then we discuss the differences between programs and scripts.

For a list of all Apple Script classes and methods supported by pro Fit, read pro Fit's dictionary from your Apple Script compiler, e.g. by choosing "Open Dictionary..." from the File menu of Apple's Script Editor.

There is an Apple Script equivalents for most commands of pro Fit's built-in compiler. Appendix C of this manual provides a cross reference between the commands of pro Fit's compiler and the corresponding operations in Apple Script.

> Apple Script is a very powerful programming language. However, it may be confusing for the beginner. The easiest way to get started is using Apple Script's "recording" capabilities. Just open the Script Editor and click the Record button. Now go into pro Fit and do (by hand) what your script is supposed to do. Script Editor records all your actions as Apple Script commands. Once you are through, go back to Script Editor and click the Stop button. Your script is now complete.

Note that this chapter is not intended to give a beginner's introduction to the Apple Script language. We will, however, explain some its aspects as we use them. To learn more about Apple Script, consult the dedicated literature, such as the "Apple Script Language Guide" distributed by Apple.

Examples

Opening and closing a single file

The following is a very simple Apple Script for opening and closing a single file:

```
tell application "pro Fit"
```

```
           open file "measured data" -- open a file
           run program "Analyze" -- analyze it
           close window "measured data" -- close it
       end tell
```

The script starts with the statement `tell application "pro Fit"` which indicates that all subsequent statements (until `end tell`) are to be sent to pro Fit. The following lines tell pro Fit to open a file called "measured data", run the program "Analyze" from the Prog menu, and then close the file again.

To use this script, you must enter it in a script editor, such as Apple's Script Editor:
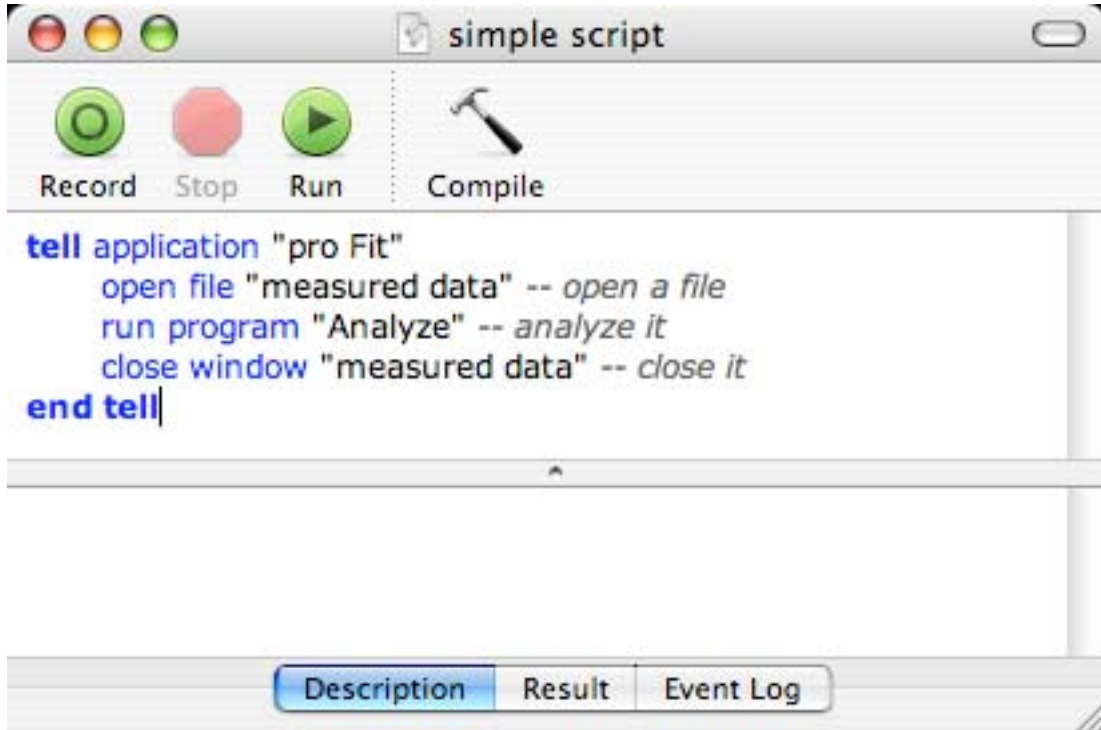
When you click Run, the script is compiled and then executed. When compiling the script, the statements are converted into Apple Events, data packets that can be exchanged between applications. When running the script, they are sent to pro Fit.

As mentioned above, you can save the script as a "Compiled Script" and then load the compiled script from pro Fit by choosing "Load Plug-in..." from the "Customize" menu. The script is added to the Prog menu from where it can be run.

Batch processing

Imagine you have a large number of data files in a folder. You want to open each of these files from pro Fit and analyze its data. Without scripting, you would have to open each file by hand, run your analysis, then close it again – boring work if you have to do it often. The following script does it all for you:

```
-- bring up a dialog for selecting the folder of the files to analyze
set myFolder to choose folder with prompt "Choose a folder with data files:"

-- create a list with all files in the folder
set myFiles to list folder myFolder -- a list of files in myFolder
```

```
set myFileCount to count myFiles -- the number of files in myFolder

-- now start working with pro Fit
tell application "pro Fit"
       set oldErrorAlerts to error alerts -- save error alert status
       set error alerts to false -- pro Fit should not show alerts
       activate -- bring pro Fit to front
       repeat with i from 1 to myFileCount  -- go through all files
              set theFile to item i of myFiles -- get the i-th file
              try
                     -- open the file for processing as data file:
                     open file ((myFolder as string) & theFile) as table
                     write line "found: " & theFile  -- write comment to Results window
                     close window theFile saving no -- close without saving
              on error errText
                     write line "cannot open: " & theFile & " (" & errText & ")"
              end try
       end repeat
       set error alerts to oldErrorAlerts -- restore
end tell
```

This script first brings up a dialog box for selecting a folder by using the Apple Script extension choose folder with prompt. Then it goes through all the files in this folder and uses the command open file *name* as table for opening the file as a data window. It also uses the command write line *text* for writing a text into the results window. Then it closes the file.

The open file and close window commands are enclosed by the statements try and on error. If any of these commands fails and returns an error, the write line statement between on error and end try is executed.

Note that we are setting a property called error alert to false before opening the files. This tells pro Fit that it should not show any error alerts of its own when it cannot open a file.

The above example simply opens each file in the folder and closes it again. In practice, you may e.g. want to run a program on each opened file. For this purpose, simply insert

```
                 run program "MyProg" -- analyze it
```
after the command open file, where "MyProg" is the name of the program you want to run.

The following is a more complete version of the above script. It not only runs a program on each opened file, it also defines the program, adds it to pro Fit's Prog menu, and then exchanges data with it:

```
-- the following defines the pro Fit program run for each data file:
set scriptProgram to ¬
       "
program ScriptProgram;
 var sum, i;
begin
 sum := 0;
 for i := 1 to nrRows do
   if DataOK(i,1) then sum := sum+data[i,1];
 globalData[1] := sum;         { store result }
end;
"
```

```
-- bring up a dialog for selecting the folder of the files to analyze
set myFolder to choose folder with prompt "Choose a folder with data files:"

-- create a list with all files in the folder
set myFiles to list folder myFolder -- a list of files in myFolder
set myFileCount to count myFiles -- the number of files in myFolder

-- now start working with pro Fit
tell application "pro Fit (ppc)"
        set oldErrorAlerts to error alerts -- save error alert status
        set error alerts to false -- pro Fit should not show alerts
        activate -- bring pro Fit to front
        compile scriptProgram -- add the above program to Prog menu
        set myTable to make new table -- open new data window
        set k to 1 -- a counter for opened files
        repeat with i from 1 to myFileCount
                set theFile to item i of myFiles -- get the i-th file
                try
                        open file ((myFolder as string) & theFile) as table -- open the file
        write line "processing: " & theFile
                        run program "ScriptProgram" -- run the program in pro Fit
                        close window theFile saving no -- close without saving
                        set sum to globalData 1 -- get result
                        set cell k of column 1 of myTable to sum -- store it in the table
                        set k to k + 1
                on error errText
                        write line "cannot process: " & theFile & " (" & errText & ")"
                end try
        end repeat
        delete program "ScriptProgram" -- remove the program from Prog menu
        set error alerts to oldErrorAlerts -- restore
end tell
```

The above script starts with the definition of the program to be run by pro Fit. Then it opens a new data window and stores its reference in "myTable". Now it opens each data file in the designated folder, calculates the sum of the values in its column 1 and stores these sums in column 1 of the data window myTable.

The script starts with
```
        set scriptProgram to ¬
```

This statement sets the symbol scriptProgram to the text following it. The symbol ¬ at the end of the line tells the script editor that more lines follow (to generate this symbol, type the return key while holding the shift key down).

The statement
```
        compile scriptProgram -- add the above program to Prog menu
```

sends this text to pro Fit and tells pro Fit to compile it, i.e. to add it to the Prog menu.

Then, the script creates a new data window using the command
```
        set myTable to make new table -- open new data window
```

The symbol myTable becomes a reference to the new data window.

Now, the files of the designated folder are opened one by one. After a file is opened, the `ScriptProgram` is run and the file is closed again. Then the script retrieves the result of the program from `globalData[1]`. The values in the array `globalData` can be accessed from scripts by using the object globalData and an index, such as

set sum to globalData 1 *-- get result*

The result retrieved in this way is transferred to the k-th row of column 1 of the data window myTable:

set cell k of column 1 of myTable to sum *-- store it in the table*

As you can see, scripts can exchange data with pro Fit, either through globalData or by accessing values in a data window.

There are other ways of interaction between scripts and pro Fit. They are explained in the last section of this chapter, which lists all Apple Script commands and objects supported by pro Fit.

When to program, when to script

As you may have realized, there are various things you can do through Apple Scripts as well as from a program defined within pro Fit. For example, you could define the following program for writing the sum of the two first cells of a data window into the results window:

```
program Sum;
begin
  Writeln(data[1,1]+data[1,2]);
end;
```

Alternatively, you could do the same from an Apple Script:

```
tell application "pro Fit"
  set sum to value of cell 1 of column 1 + value of cell 1 of column 2
  write line sum
end tell
```

Even though the above examples do the same, you will prefer the program, because defining programs is usually more convenient and faster.

In practice, you probably use programs more often than Apple Scripts. Programs can be defined within pro Fit, they are much faster, and they are better suited for numerical applications. However, there are some things that you simply cannot do from a program, such as exchanging data with other applications, communicating with the Finder, batch processing a large number of files, etc. For these tasks, you can use Apple Scripts.

You can combine the advantages of Apple Scripts and programs: To call an Apple Script from a program, first add it to the Prog menu (choose Load Plug-in... from the Customize menu), then call it with CallProgram(…). To call a program from an Apple Script, compile it and use the command run program.

Apple Script methods and classes

The following describes some applications of the most important Apple Script methods and classes supported by pro Fit. For a complete overview, check pro Fit's dictionary and have a look at Appendix C of this handbook.

Some methods

open: Open the specified object(s)
     open file -- list of objects to open
        [as data window/drawing window/funcProg window/text window]

If the file is a file of type text, you can indicate if it is to be opened `as text` (i. e. in a new function window) or `as table` (i. e. in a new data window).

Examples:
open file "HD:myData" -- opens the file "myData" on the disk HD
open file "data" as data window -- opens the (text) file "data" as data window
open file "Drawing1" -- opens the file called "Drawing" in pro Fit folder

calculate statistics: performs statistical calculations on the given window
     calculate statistics reference -- the data window for the statistical analysis
        [column integer] -- the column for the statistical analysis. 0 to use all columns.
        median boolean -- set to true to calculate median, minimum, maximum.
        basic boolean -- set to true to calculate basic statistical information.
        skewness boolean -- set to true to calculate Skewness and Kurtosis.
        [selected cells boolean] -- true if statistical analysis of the current selection
        [selected rows boolean] -- true if statistical analysis must be applied
                                -- only to the data contained in the selected rows.

To retrieve the results, use e.g.
        get statMean of results -- returns the mean value
(Available selectors of results are listed for class "calculation results" below)

capture: Switches capturing on and off
     capture constant -- to file | enabled | disabled | done
        [to alias] -- the file to capture into (not used for options on | off | done)
     Example:
          tell application "pro Fit PPC"
             capture to file "HD:logFile" -- *start capturing to logFile*
             write line "hi there" -- *will be captured*
             capture disabled -- *disable capturing temporarily*
             write line "some text" -- *will not be captured*
             capture enabled -- *enable capturing*
             write line "add this to log file" -- *will be captured*
             capture done -- *close the capture file*
          end tell

close: Close a window
     close reference -- the window to close
        [saving yes/no/ask] -- Specifies whether or not changes should be saved

Windows can be specified by name or index (1 is the frontmost window, 2 is the window behind the frontmost window).

If you append the specification saving yes then all changes are saved – if the window has not yet been saved to a file, you are asked to specify where you want to save the changes. If you append saving no, changes are not saved. If you append saving ask or if you do not append a saving specification and the window contains unsaved changes, pro Fit will ask if you want to save the changes.

Examples:
```
close front window   -- prompts for saving unsaved changes
close window "Table1"  saving no  -- closes the window without saving
```

compile: Compile a function or program written in pro Fit's definition language.
```
compile  reference  -- text or file to compile
```

Examples:
```
compile file "HD:myProg"     -- compiles the given file
compile "function lin; begin y:=a[1]*x; end;" -- compiles a text
```

the following is a more realistic way to define and compile a larger program from a script: (note that you can create the character "¬" by hitting option-return – this character specifies that the line is continued on the next one):
```
set myProg to ¬
"
program test;
 var i, sum;
 begin
   sum := 0;
   for i := 1 to nrRows do
    if dataOK(i, 1) then sum := sum + data[i,1];
   writeln('sum of col 1: ', sum);
 end; "

tell application "pro Fit"
  compile myProg       -- compiles the above definition
  run program "test"  -- and runs it
end tell
```

delete: Removes a function or program from pro Fit's menus
```
delete  reference  -- The function or program to delete
delete program "FitFrontWindow"     -- deletes the specified program
delete function "linear"     -- deletes the specified function
```

do script: Compile and execute one or more Pascal statements
```
do script  reference  -- the window or the statements to execute
```

Examples:
```
do script "Writeln('Hello world');" -- executes the pascal statement
```

make: Make a new window or shape.
```
make
        new:  type class  -- the class of the new element: 'table', 'drawingWindow',
```

                        -- 'textWindow'
       [with properties: record] -- the initial values for the properties of the element
Result: reference -- to the new object(s)

The keyword "new" is optional in Apple Script.

what specifies the type of window to be opened. Specify `table` for a data window, `drawingWindow` for a drawing window, `textWindow` for a function window.

The `with properties` parameter specifies the properties of the window in an Apple Script record. All types of pro Fit windows have the property `name` holding the name of the window as a string. In addition to this, data windows have the properties `nrRows` and `nrCols` with the numbers of rows and columns.

Examples:
make table with properties {name:"myTable"}
           -- creates a new data window having the name "myTable"
make drawingWindow with properties {name: "lookatthis"}
           -- creates a new drawing window having the name lookatthis
make textWindow   *-- creates a new function window*
make table with properties {name:"small", nrCols:10, nrRows:20}
           -- creates a data window with name "small", 10 columns and
           -- 20 rows

-- the following creates a new data window and then closes it using a
-- temporary reference to the window
set myRef to make new table
close myRef

save: Save a window
    save reference -- the window to save
       [in alias] -- the file in which to save the object
       [as data file/drawing file/EPS file/function file/PICT file/text file]
          -- file type for data export
Windows can be specified by name or index (1 is the frontmost window, 2 is the window behind it, etc). Note that Apple Script allows you to specify indexed objects in various ways (such as `window 1, front window, 3rd window, last window`)

Optionally, you can specify the file where the window is to by saved after `in`. If you do not specify the file where to save the window and the window has never been saved before, you are prompted to enter a file name. If you don't specify the file where to save the window and the window has been saved before, the window is saved to the same file as before.

If the specified window is a data window, it is saved as a regular pro Fit file by default (this is equivalent to specifying "as table"). If you want the data window to be saved as text file for exporting it, specify "as text".

Examples:
save window 1 to file "HD:data"   *-- saves front window to file*
save window "data" to file "data.txt" as text   *-- saves as a text file*

12 Printing

There is a wide range of different printers that can be connected to a Mac OS machine, and each of these printers have different capabilities, resolutions and command languages. pro Fit allows you to get the best out of most of the commonly used printers if you follow the guidelines described in this chapter.

Basically, there are two possibilities for printing pro Fit drawings. You can print from pro Fit directly (using the **Print** command from the File menu) or you can export a drawing to another application, such as a word processor (using the **Copy** or the **Create Publisher** commands or by dragging it to the other application), and print it from there. The next two sections discuss these two possibilities separately.

Printing from pro Fit

Before printing, you should choose **Page Setup** from the File menu.

You can print the active window by selecting the **Print** command from the File menu.

For drawing windows, pro Fit offers two different modes of printing: *Quartz* and *Postscript*. You can select the desired method by choosing Preferences... from the pro Fit menu. In the dialog box that comes up, click the icon "Printing" in the list at the top..

If **Quartz** is checked, pro Fit prints drawings using the Quartz engine introduced with MacOS X. This provides best results on all printers as well as for pdf generation.

If **PostScript** is checked, pro Fit generates PostScript commands while printing a drawing window. On a PostScript printer, this provides good results. However, printing text with PostScript may sometimes lead to problems on some printer drivers.

Do not use Postscript when printing to a non-PostScript printer.

Note 1: Bitmap based patterns appear as gray levels under PostScript.

Note 2: If you experience postscript problems when drawing plots containing a large number of points, try to increase the number of points drawn with one stroke in the Plotting panel of the Preferences command.

In the printing section of the Preferences dialog, you can also choose the default page setup to be used for the various window types within pro Fit.

Printing a pro Fit drawing from another application

When drag or copy pictures to other applications, pro Fit can encode these pictures in two different ways: PICT and PDF. PICT is a legacy format dating from the times prior to MacOS X and has limited capabilities, but it has the advantage that it is understood by nearly all applications. PDF is a more modern format with much improved capabilities, but not all applications are able to handle it. For example, Microsoft Word up to version 2003 does not support importing PDF images through the clipboard.

In the "Copy" section of pro Fit's Preferences command, you can specify if pro Fit is to export one or both of these formats. :
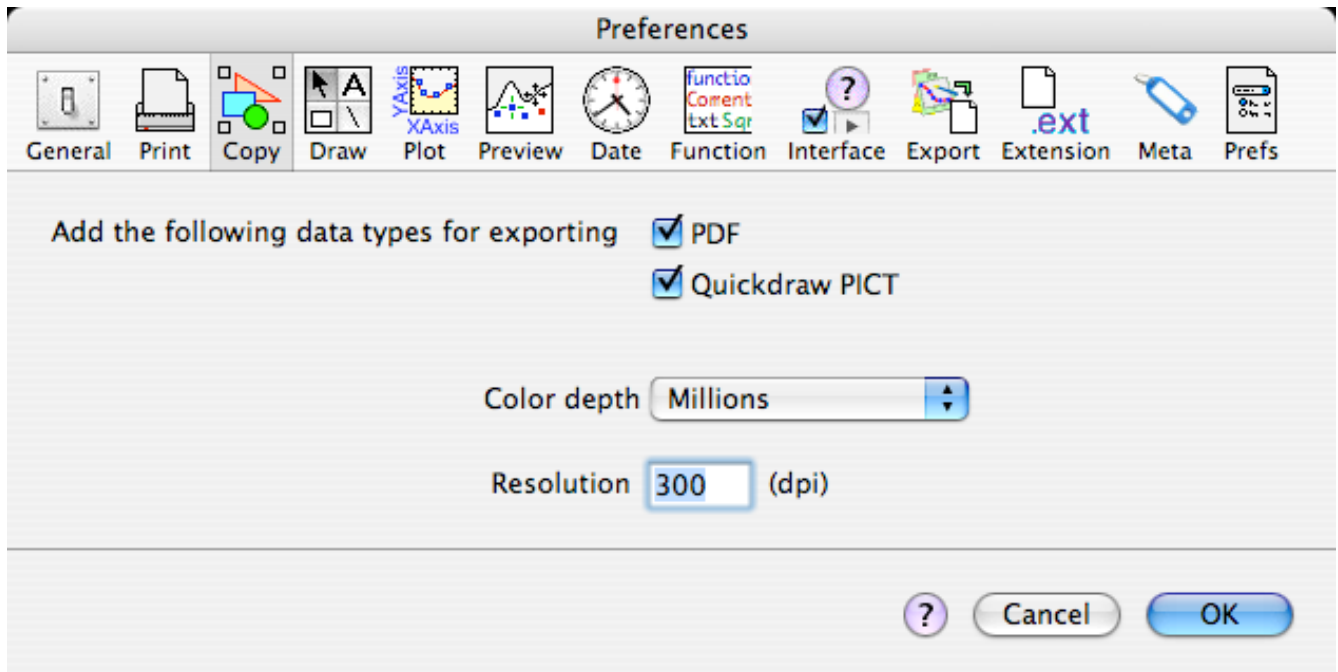
PDF: Many MacOS X applications support the exchange of PDF data over the clipboard. This is the preferred format to export pictures from pro Fit because PDF data is resolution independent and allows excellent rendering on nearly all devices.

Quickdraw PICT: Some legacy or monopolist applications do not support PDF over the clipboard. For these applications, pro Fit can place Quickdraw pictures on the clipboard. These are accepted by nearly all applications. But they are not resolution independent. For best portability, pro Fit exports bitmap based pictures. They should render fine on all devices as long as the resolution of the bitmap is equal to or much higher than the one of the device. Therefore, when using Quickdraw pictures, you must set the resolution correctly. The field **Resolution** gives the resolution of the bitmap in dots per inch. The larger the resolution, the finer the bitmap will print and the more memory it will use. Set this field to your printer resolution or to an integer divider of your printer resolution. The pop-up **Color depth** sets the number of bits for encoding the color of each pixel.

Note: In most situations, you can check PDF as well as Quickdraw PICT. The target application will usually select whatever information it handles best. Uncheck one of these options if you want to force the target application to use a given format (if it can).

13   Preferences

pro Fit offers many possibilities to customize its features: You can choose the format for exporting pictures, the preferred method for printing, you can save your preferred user interface options, etc. All of these settings are saved in pro Fit's preferences file. During start-up pro Fit looks in the Preferences folder of the System folder for its preferences file. If the file is there, pro Fit reads its standard settings from it. If the file is not there, pro Fit creates a new preferences file. You can switch to another preferences file or create a new preferences file anytime later.

If you do not want to load the standard preferences file in the system folder, hold down the option and the shift key while starting up pro Fit.

Most of pro Fit's settings are controlled by choosing Preferences... from the pro Fit menu. Doing so brings up a dialog box with several *panels*. Each panel controls a set of options. To choose a panel, select its icon from the list at the top of the dialog box.

The panels **Printing** and **Copy** are discussed in Chapter 12, "Printing".

We will not provide a detailed discussion of the other panels in this handbook since each of them comes with its own help button to the left of the Cancel button:
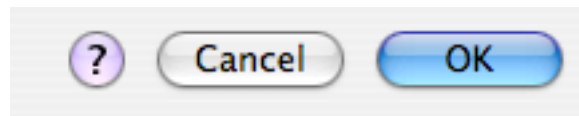
Clicking the help button brings up a help screen that explains the functions of all the settings.

14   General features

Getting help

pro Fit offers a powerful on-line help based on Apple's Help Viewer application. The pro Fit Help can be accessed by choosing **proFit Help** from the help menu, or by clicking one of the question marks that you find in pro Fit windows and dialog boxes. Balloon help is also supported.

When defining functions and programs there is a special feature based on a dedicated help menu which is always present in the header of function windows. See chapter 9, "Defining functions and programs", for more information on this help menu.

The on-line help system for pro Fit 5.6 is embedded into the pro Fit application bundle and therefore usually not visible to the user.

Help tags

pro Fit supports help tags for providing information on individual user elements.

Help tags on MacOS X are displayed automatically when you keep the mouse over an item of interest for at least half a second.

On-line evaluation of mathematical expressions

Wherever pro Fit expects a numerical input, such as in spreadsheets or dialog boxes, you can enter a mathematical expression. For example, instead of typing a number directly, you can use a mathematical expression like "exp(1)" or "6+sin(pi/4)". pro Fit reads the mathematical expression you typed or pasted and calculates the numerical result.

Text windows, such as the result window, can be used as a calculator by typing an expression on a new line, positioning the insertion point on that line, and hitting the Enter key. The result is displayed on the next line.
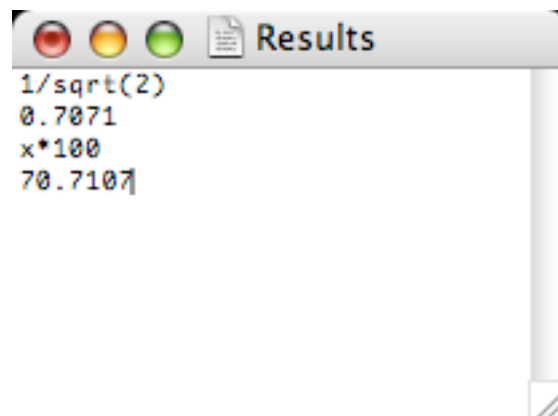
You can also use mathematical expressions in all pro Fit dialog boxes. As an example, if you want to tabulate a function between 0 and two times `pi` at intervals of `pi/5`, type command-T and enter the following:

Table of Polynomial(x)

Tabulate by varying   x

Min: 0   Max: 2*pi

Step: π/5   (0.5, 'auto', 'points')

☐ Use fitted parameters

(?)   Cancel   OK
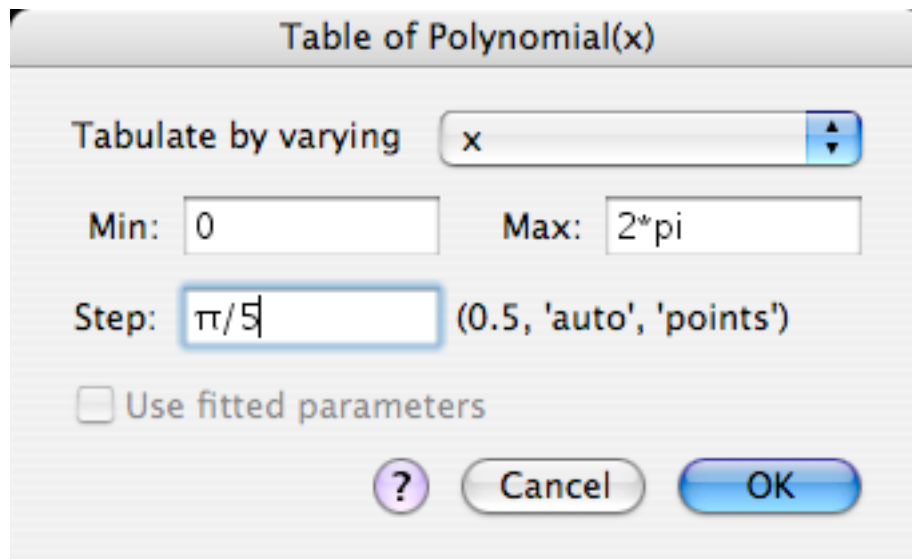
When typing a mathematical expression, you use the same syntax elements that are available when writing a function definition. In on-line mathematical expressions, `x` is equal to the last result that was evaluated, and `a[i]` is equal to the input values shown in the current parameters window. You can use all the predefined functions available when writing the definition of a function. As an example, after a successful fit you can type 'ChiSquared' in a data window cell. This tells pro Fit to set the value of that cell to the mean deviation obtained in the last fit (see Chapters 9 and 10, together with pro Fit's on-line help, for more information on predefined functions).

Let's look at a simple example that illustrates how you can use pro Fit's understanding of mathematical expressions when you are pasting into a data window. Write the following text and copy it to the clipboard:

a[2] → fittedParams(2) → a[2] – fittedParams(2) → paramSD(2)
a[3] → fittedParams(3) → a[3] – fittedParams(3) → paramSD(3)
a[4] → fittedParams(4) → a[4] – fittedParams(4) → paramSD(4)

Where the '→' stands for a tabulator character and each line is terminated with a carriage return (¶). If you paste the above text into a data window after a successful fit, you automatically obtain a table containing the parameter values before the fit, the values after the fit, their difference, and the resulting standard deviations.

File info

pro Fit lets you save a comment with every one of its files. You can edit this comment with the **Get Info** command from the **File** menu. Choosing Get Info presents a dialog box with a large field for editing text.

Infos about "Table 1"

These poitns were recorded using Finac's correction. Colums 3 and 4 give the temperatures at the end points.

3/5/04 TT

Cancel   OK
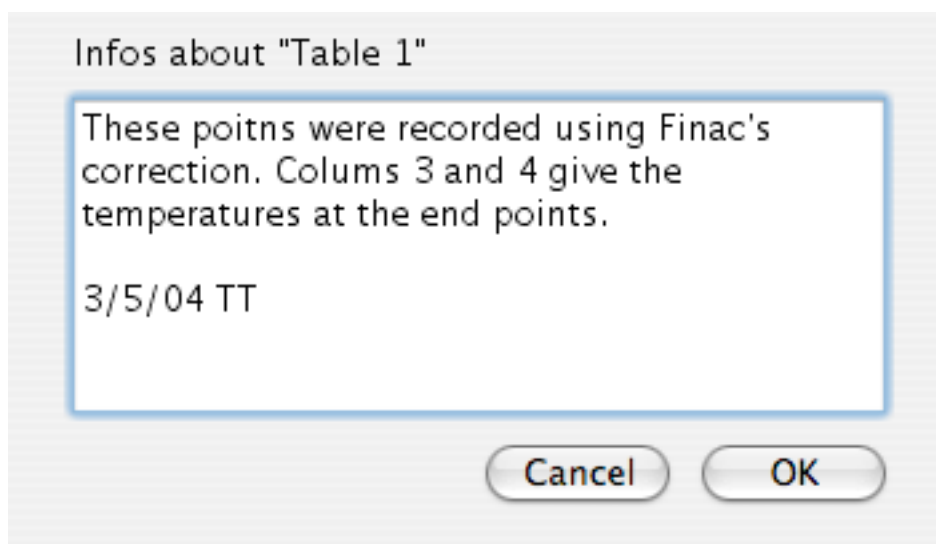
You can add an info comment to data windows, drawing windows and functions or programs.

The **data windows** let you view and edit this information directly, without using the Get Info command. For this you drag down the info hook (a black area on top of the right scroll bar) of a data window to create an info field of the desired size. See Chapter 4, "Working with data" for more information on data windows.

Table 1

These poitns were recorded using Finac's correction. Colums 3 and 4 give the temperatures at the end points.

3/5/04 TT

Index	x-Value	y-Value	t1	t2
1	0.00000	0.00000		
2	0.62832	0.39478		
3	1.25664	1.57914		
4	1.88496	3.55306		
5	2.51327	6.31655		
6	3.14159	9.86960		
7	3.76991	14.21223		
8	4.39823	19.34442		
9	5.02655	25.26619		
10	5.65487	31.97752		
11	6.28319	39.47842		
12				
13				

22, 10
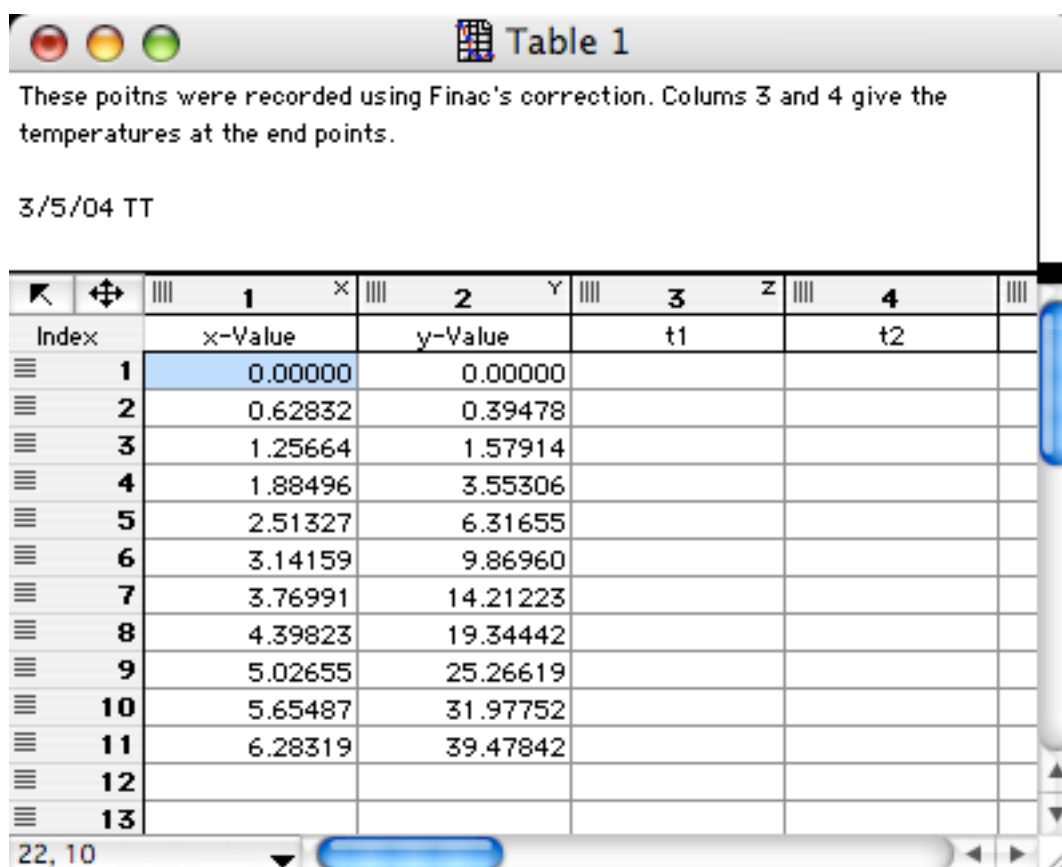
Note that the info comments are in general only saved in files that have proFit's standard formats. If you save a function definition as normal text files (TEXT format) or if you save a drawing window as a

picture or EPS file (PICT format, EPSF format), the info comments are not saved. If you save a data file as TEXT, you have the option of placing the info comments right at the beginning of the text file, as a header. To set this option, you have to choose "Custom format" in the dialog box that comes up when saving text files.

Find and Replace

proFit provides Find & Replace features to help you navigate through text or data. This feature is available for the results window and all function windows as well as for all data windows. You will find it useful when you are editing the definition of a function or a program inside a function window.

The Find & Replace commands are found in the Edit menu:
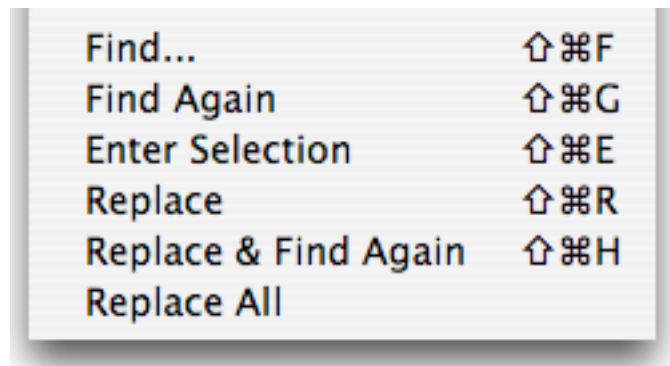
When you choose **Find...** for a text window, the following dialog box appears:
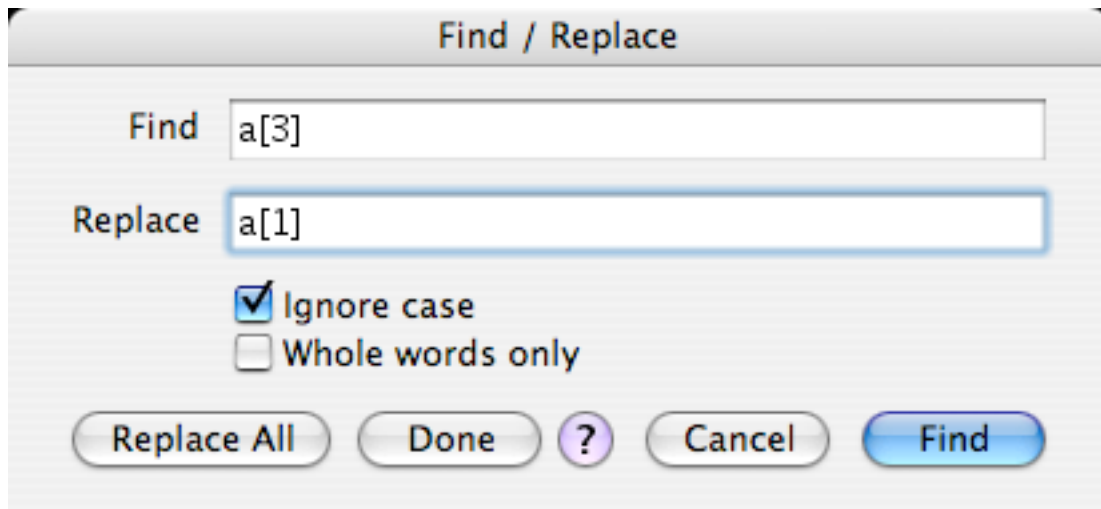
Type the text you are searching for and the replacement text in the **Find** and **Replace** edit fields. Use the radio buttons **Forward/Backward** to start the search by moving down from the current insertion point towards the end of your text, or up towards the beginning. Click the **Find** button to start a search, click **Done** if you don't want to start a search yet. Click **Replace All** to replace all the occurrences of the text appearing in the Find item with the text appearing in the Replace item.

Use the menu command **Enter Selection** to enter the currently selected text in the Find field of the Find&Replace dialog box. Choose **Find Again** to restart a search (the fastest way to find all occurrences of a text is to select it and choose Enter Selection and Find Again in rapid succession). Use

Replace to replace the current selection with the text in the Replace field of the Find&Replace dialog box. **Replace and Find Again** combines the last two commands. **Replace All** is equivalent to the Replace All button in the Find dialog box.

Note that by using the Enter Selection command, or by copying some text and pasting it into the Find and the Replace field, you can enter text that you cannot enter by typing in the dialog box, such as carriage returns (¶) and tabs (→).

When you choose **Find...** for a data window, the following dialog box appears:
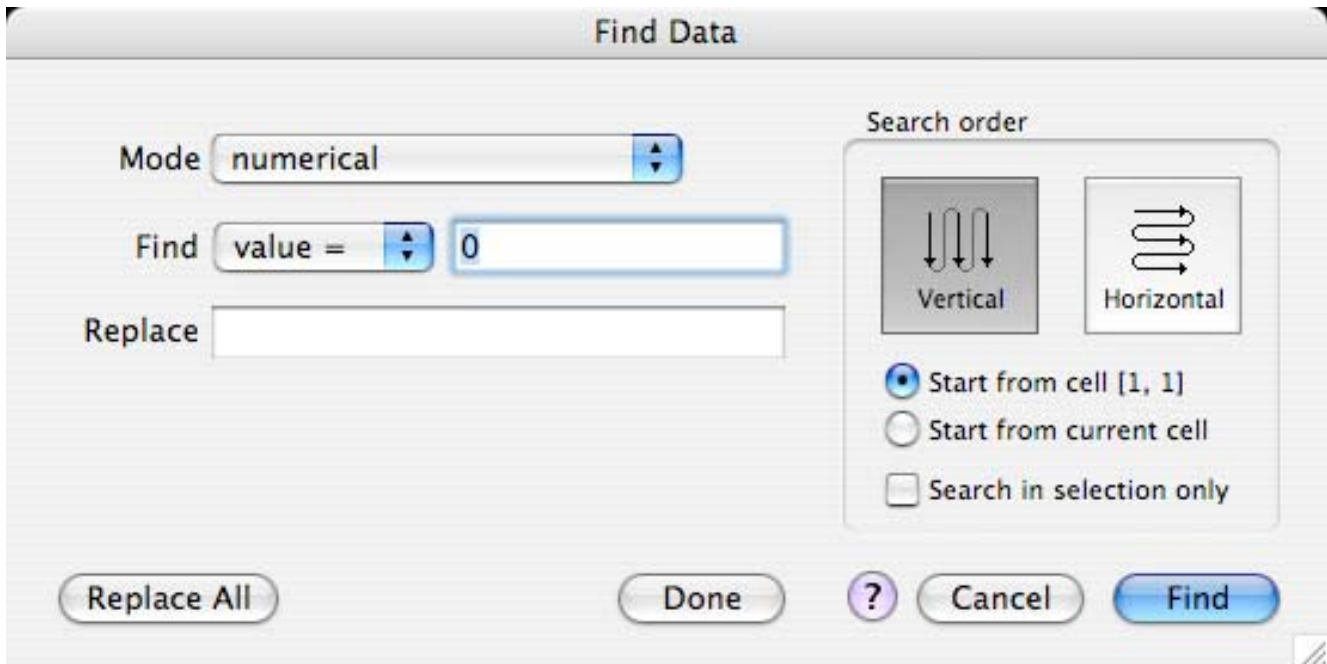
The pop-up **Mode** specifies if you are searching for numeric matches or for textual matches.

- Select "numerical" for using a numeric comparison, where e.g. 1.0 equals 1.0000. This search mode also allows you to search for values that are larger or smaller than a given threshold.

- Select "textual" for using a character by character comparison, where e.g. 1.0 does not equal 1.0000. This search mode also allows you to search for cells containing a given substring or not contaning a given substring.

Under **Find**, you specify the search criterium in the pop-up and the value/string to be searched for in the edit field.

Under **Replace**, you specify the value/string to be used when replacing found cells.

In the group **Search order** you specify the order in which the cells are searched and what cells are searched.

Select **Vertical** for searching column by column, from left to right. Select **Horizontal** for searching row by row, from top to bottom.

Check **Start from cell [1,1]** to start your search with the first cell of the window. Check Start from current cell to start the search from the first currently selected cell.

Check **Search in selection only** to search within the selected cells only, uncheck it to search in the whole window.

Click **Replace** All to replace each cell matching the Find criterium.

Click **Done** to dismiss the dialog box but retain its settings.

Click **Cancel** to dismiss the dialog box without retaining its settings.

Click **Find** to find the next match.

Note: The Info field of the data window, the column names and the column and row numbers are not searched.

Contextual menus

Some of pro Fit's windows allow you to use "contextual menus". To invoke a contextual menu, click on a desired part of pro Fit while holding down the control key.
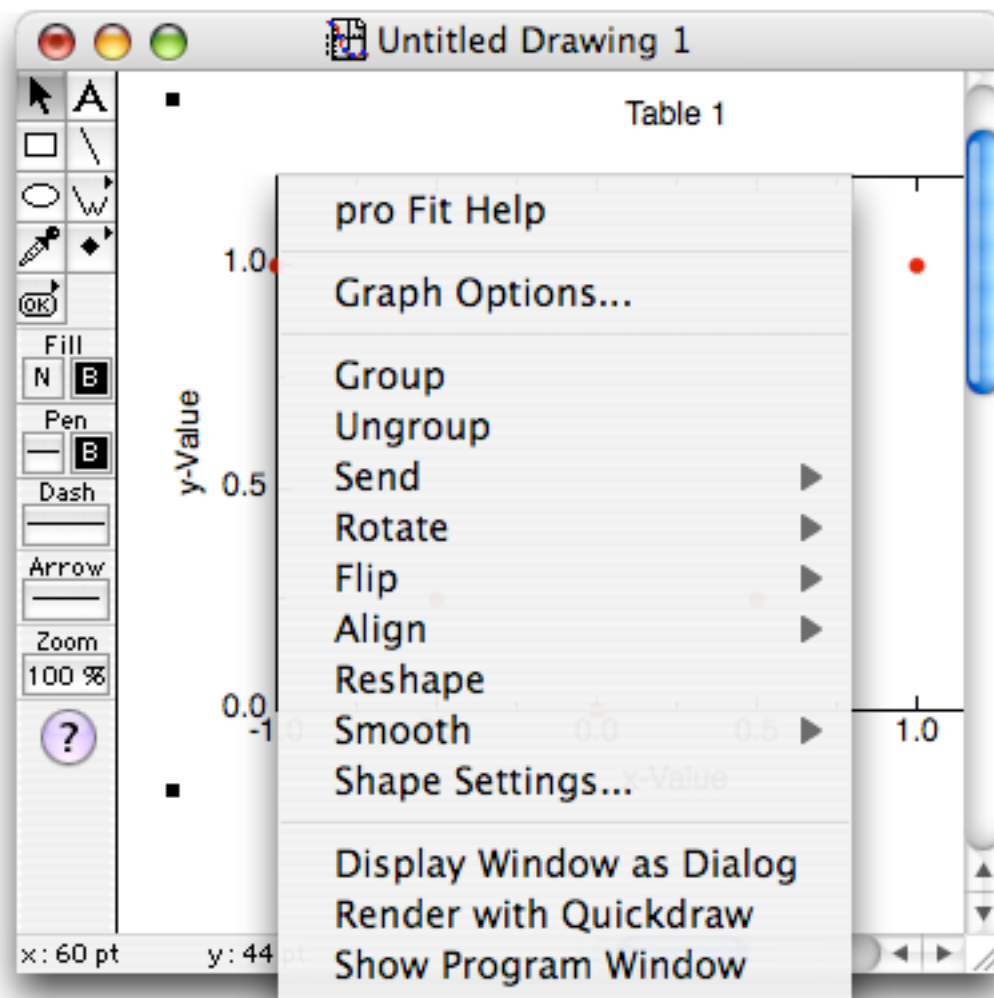
Shortcuts and other options

Although most of pro Fit's features and commands are readily accessed through its menus, there are some more advanced or rarely used features that require the use of modifier keys like the option key, the command key, or the shift key.

This is a short list of these features:

action	modifier keys
• Selecting a tool in the tools palette of drawing windows	**option** to keep the tool selected after drawing the corresponding object.
• Dragging objects in drawing windows	**command** to constrain the movement along 45° lines. **shift** to constrain the movement to horizontal and vertical directions. **option** to duplicate an object instead of simply moving it.
• Drawings objects in drawing windows	**option** or **shift** to get a square bounding box.
• Drawing lines in drawing windows	**shift** to make the line horizontal, vertical or diagonal (at 45°) **option** to make a diagonal line
• Drawing polygons in drawing windows	**option, shift** same as for lines **command** double-click to produce a corner that remains a corner even when the polygon is smoothed.
• Resizing objects in drawing windows	**option** to keep the bounding box of the object square (height=width). **shift** to maintain the horizontal and vertical proportions of the object, its height, or its width. **command** to resize the size of texts in a group.
• Resizing lines in drawing windows	**option** to get a line constrained to 45° directions. **shift** to maintain the direction of the original line, or to make the line vertical or horizontal
• Clicking objects in drawing windows	**shift** to select an object without de-selecting other already selected objects.
• Clicking graphs in drawing windows	**option & command** + click to see the plot coordinates of the point you are indicating with the cursor. **option & command** click, and then press **shift**

		to select an area of the graph to be enlarged.
		command double-click
		to make a graph the 'current graph'.
		command & shift double-click
		to remove the 'current graph' setting.
•	Clicking nothing in drawing windows	**command** + click
		to zoom in, centering the clicked point in the new view.
		option & command click,
		to zoom out.
•	Using the line style pop-up menu in a drawing window to change the line styles in a legend	**shift**
		to change the line styles of all the lines in the legend.
		option
		to change the line style and set the attribute 'points connected' for the data plot in the first row of the legend.
		shift & **option**
		to change the line styles of all the lines in the legend and set 'points connected' for all data plots.
•	Using the point style pop-up menu in a drawing window to change the point styles in a legend	**shift**
		to change the point style of all the data plots in the legend.
•	Clicking a cell in a data window	**option**
		to select the whole column above the clicked cell.
		shift
		to enlarge a selection.
•	Clicking the column number cell in a data window	**command**
		to set the default columns (x, y, Δx, Δy) using a pop-up menu.
•	Clicking on the 'larger font size' controls in the text-edit dialog box	**option**
		to increase the font size by 1 pt only .
•	Clicking on the 'subscript/superscript position' controls in the text-edit dialog box	**option**
		to change the vertical position of the selected text by 1 pt only.
•	Choosing 'New Function' from the file menu	**option**
		to open a new definition window containing a sample function definition.
		option/shift
		to open a new definition window containing a sample program definition.
•	Importing text files	**option**
		to tell pro Fit not to ask for information and to open the text files as data files using the current settings.
•	Saving a drawing as an EPS file.	**option**
		to create a TEXT file containing the PostScript information.

- Using lists in dialog boxes (e.g. the y-column list in the plot data dialog box).

  shift click**, shift** and drag
  to select more than one item.
  shift click
  to de-select an item.

- Clicking with the lens tool in the Preview Window

  command to drag a selection rectangle specifying the region to enlarge.
  option to zoom out instead of zooming in

- Selecting an item from the Help menu in a Function window

  option
  to paste the template with a ';' and a carriage return
  command
  to enable pasting templates and disable help panels
  shift
  to enable help panels and disable pasting templates

- Clicking a marker in the Preview window

  option
  to transform the clicked marker into the reference marker

- Moving a marker with the arrow keys in the Preview window

  option
  to let the marker go outside the ranges of the preview.

- Using the left and right arrow keys in a data window

  option
  to move the insertion point by one character within the active data cell.

- Clicking in the data window

  command
  to create a discontinuous selection

- Starting pro Fit

  option and shift
  in order not to load the standard preferences file

Another commonly used shortcut is typing a period ('.') while holding down the command key. This is equivalent to typing the escape key and it interrupts most of the calculations. Use it to stop the plotting of a function, to stop fitting, to cancel printing, or to interrupt lengthy calculations.

The combination Command-key/period is also interpreted as typing 'Cancel' in dialog boxes. The escape character is also interpreted as 'Cancel'. Return or Enter are always interpreted as clicking the outlined button.

Appendix A: About numbers

Floating point numbers

pro Fit uses three different formats for representing floating point numbers (or float):
- real (or float): This format has smallest accuracy but requires minimum size. It is used in data windows if you set the range of a column to "–1E30 ... 1E30".
- double: This format has better accuracy but requires more size. It is used in data windows if you set the range of a column to "–1E300 ... 1E300".
- extended (or native double): This is the format used for internal calculations. It has the same or better accuracy as the double format.

The following list summarizes the features of each data type for the Power Macintosh and the 68k version of pro Fit:

	real	double	extended (native double) Power Mac	$68k^{\dagger\dagger}$
minimum negative number	–3.4E38	–1.8E308	–1.8E308	–1.1E4932
maximum negative number	–1.2E-38	–2.2E-308	–2.2E-308	–1.7E-4932
minimum positive number	1.2E-38	2.2E308	2.2E308	1.7E-4932
maximum positive number	3.4E38	1.8E308	1.8E308	1.1E4932
decimal digits	7–8	15–16	15–16	19–20
size (bytes)	4	8	8	$10/12^{\dagger}$

$\dagger$ The FPU version uses 12 bytes, the non-FPU version 10.

$\dagger\dagger$ 68k applies to pro Fit 5.1 or earlier – pro Fit 5.5 and later only support PowerMac data.

Apart from the values in the list above, pro Fit knows four other numbers: 0, +INF (infinity), -INF (–infinity), NAN. The first three of them will do what you expect them to do. E.g. 1/0 = +INF, INF/3 = INF etc. NAN (Not A Number) is the result of any computation that cannot be carried out, such as sqrt(-1). The occurrence of NAN values in computations is reported as a run-time error.

Date and Time data

pro Fit understands and works with time data, *i.e.* absoulte calendar dates and relative time.

The Mac OS stores dates as the number of seconds since January 1, 1904 (for the technically minded, the date is stored as a long integer number, 8 byte long).

pro Fit uses the same convention as the Mac OS to store dates, but uses "double" floating point values instead of integers. With this number representation, pro Fit can store and recognize dates with second precisions until up to 10^{15} (this corresponds more or less to a 6 byte long integer) seconds after January 1, 1904. This means that pro Fit can store dates with second-precision up to **31 million years in the future**, and it can store dates with day-of-the-week precision up to 3.1 **billion** years (3×10^9 years) in the future.

Up to about 29000 years into the future, pro Fit can store dates with a precision of milliseconds, while it can store dates in the present with a precision of approximately a microsecond.

Appendix B: File formats

This appendix describes the file formats used by pro Fit for transferring data or drawings to and from other applications.

Data

The default text format

To exchange data between pro Fit and other applications, text files are used. Usually, such files hold one or more lines of text. Each line contains all values of a row separated by "tabs" ($\rightarrow$). The lines are separated by "carriage returns" (¶). It is possible to use other characters instead of tabs and carriage returns (see below).

There are two standard formats of data text files produced by pro Fit:

The *standard* format *with titles* is defined as follows:

1st line: name1 $\rightarrow$ name2 $\rightarrow$ name3 ¶
2nd line: 0.123 $\rightarrow$ 1.732 $\rightarrow$ 1.122 ¶
3rd line: 2.233 $\rightarrow$ 2.125 $\rightarrow$ 2.126 ¶
.....

The *standard* format *without titles* is very similar, but without the column titles line.

There is an interesting exception for data text files to be loaded into pro Fit. If the first line is a single star (*) pro Fit reads the second line as the column titles even if the file is loaded as being standard format without titles.

Lines are separated by carriage returns (`(char)(13)` or `'\r'` for C programmers, `chr(13)` for Pascal programmers).

The first line with the column titles is optional. These names are separated by tabs (character code 9, here denoted as '$\rightarrow$' – `(char)(9)` or `'\t'` for C programmers , `chr(9)` for Pascal programmers). If pro Fit reads a file without column titles, it sets the columns names to "Column 1", "Column 2" etc.

A typical Pascal program for writing such a file would be:

```
var out:text;
...
rewrite(text,'filename');
writeln(text,'x',chr(9),'y');
writeln(text,'1.234',chr(9),'2.341');
writeln(text,'-1.244',chr(9),'3.412');
...
close(text);
```

Some applications produce data text files using other formats, or read data text files only when they are in special formats. pro Fit provides options to read and write text files in other formats as well. The details are given in the next section.

Importing text files

For reading text files, choose Open... from the File menu, choose "Text Files" from the View pop-up and select the file to be read. You will be prompted for the following information:

If you select **Import to text window**, the file is opened as a non-data text file and loaded into a new function window.

If you select **Import to data window**, the file is opened as a data file and loaded into a data window. In this case you can select either one of the standard formats or the custom format.
  - If "Custom" format is not selected, proFit uses an intelligent translation algorithm, which recognizes most data file formats automatically. By selecting the "With Titles" format pro Fit interprets the first line in the text file as the column titles.
  - If "Custom" format is selected, you can specify the file format yourself.

Check **Header** if the input data has header bytes that are neither column titles nor data. In that case, you can specify if the header bytes are to be copied to the info field of the data window or not.

Check **Column** titles if the input data provides column titles.

Under the heading **Lines**, you can specify the formatting of the lines (rows). Either specify a delimiter that ends the line, or a maximum number of values per line. If you specify both, the line is supposed to end either after the given number of values or when the delimiter is encountered.

Under the heading **Values**, you can specify the formatting of the values (columns). The values can either all have the same format or (if you have specified a number of values per line under the heading "Lines") individual format. In the latter case, edit the format for each value individually by choosing the appropriate item in the popop Edit format for Value.

The value format specifies the type (automatic, number or text, wherein automatic specifies that after importing, any column that contains purely numerical data is converted to a number column while all other columns are supposed to be text). The length of each value in the input data is either specified by a delimiter, or by a fixed length of bytes, or as being a pascal string (i.e. a first byte specifying the number of characters and a corresponding number of character bytes following).

The section **Input** allows you to display the contents of the input file in either textual or hexadecimal representation.

The section **Output** lets you preview how the resulting data window will look like. To update that section, click the button Update.

Saving text files

You can also save data into text files in a custom format. To do this, choose Save as... from the File menu for your data window and choose "Text File" from the Format pop-up. In the dialog box that appears, select Custom format:
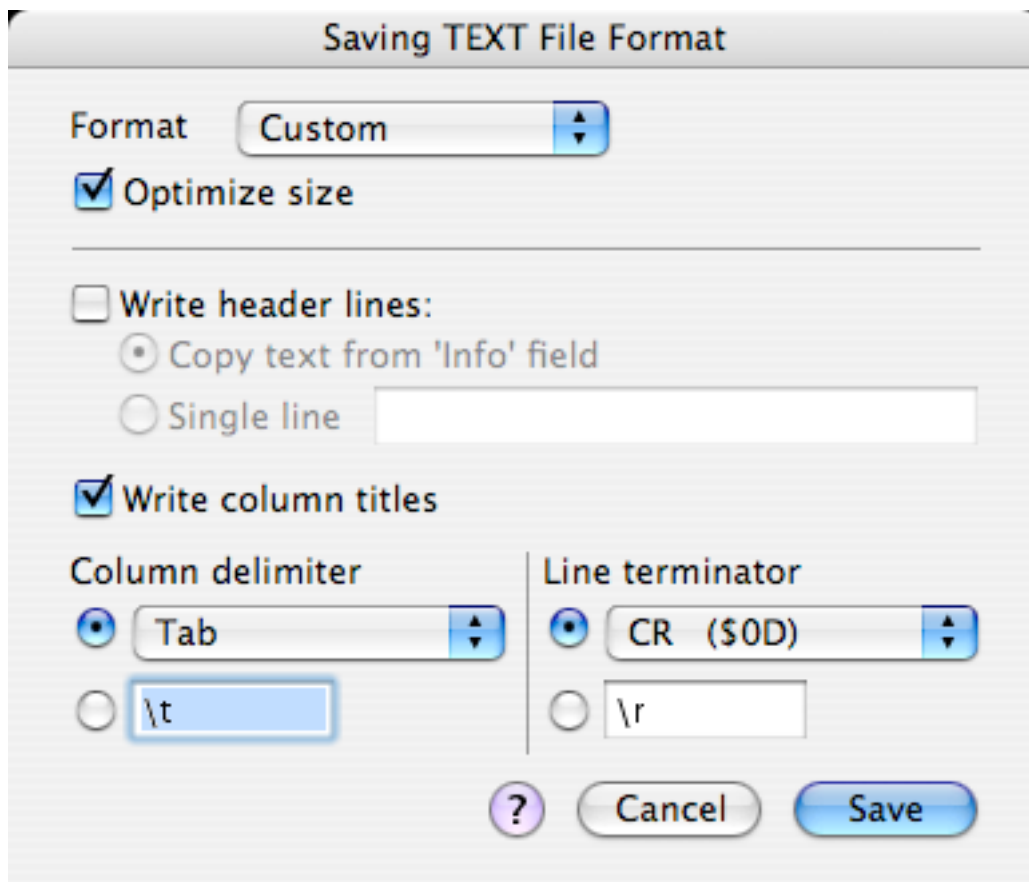
This dialog box is very similar to the one for loading text files. Again, you can select the **Column delimiter** and the **Line terminator**.

If **Write header lines** is checked, you have the option to either write a single first line with the text specified in the edit field to the right, or to copy the whole text contained in the info field of the data window as the header of the file.

If **Write column titles** is checked, the next line contains the column names separated by the column delimiter.

Check **Optimize size** to write the numbers with as few characters as possible, without loosing precision.

The native data format

If you want to exchange binary data with pro Fit, you can use pro Fit's native file format. A description of this format is given in the technical note "pro Fit file formats" in the folder "Notes" that comes with the pro Fit package.

Drawings

There are various ways to export pro Fit drawings to other applications, and pro Fit provides several image formats to do so.

You can export data through the clipboard, by drag-and-drop as well as by means of files:

Exporting through the clipboard by means of copy and paste is the most traditional way. When you copy a drawing to the clipboard, pro Fit provides picture and pdf data for the target application to use (see below).

Exporting by drag-and-drop may be more convenient in some situations. When you export by means of drag-and-drop, pro Fit also provides picture and pdf data for the target application to use (see below).

Exporting by files requires you to save the drawing window using a custom format. This is more cumbersome but allows you to use a variety of standard file formats, such as pdf, tiff, gif, png and jpg. To export a drawing in a file, choose Save As from the File menu, choose the desired file format from the Format pop-up, and select the appropriate options by clicking the button Options.

Image formats

The image formats supported by pro Fit are:

Pictures (PICT format): This is the classic format for images under MacOS. It provides compatibility with a large number of applications, but does not support all modern imaging features. In particular, it does not support transperancy and poorly supports rotated text or unicode text. pro Fit allows you to set various options for exporting pictures. The default options can be set in the PICT Options panel of the Preferences command (pro Fit menu), the options for exporting into a file can be accessed throgh the button Options in the Save As dialog box.

PDF (Portable Document Format): This is a very powerful imaging format recognized by a large number of applications. pro Fit exports PDF files for saving drawings, but it also supports the exchange of PDF data through the clipboard.

EPS (encapsulated postscript): This file format is also very common, and it can e.g. be used for exporting drawings to LaTeX editors. Note: pro Fit does not support unicode text in EPS files, but only the ISOLatin1 character set. pro Fit generates Postscript level 2 code.

JPEG (Joint Photographic Experts Group): This is a very common file format for exchanging images. It is, however, lossy, and not well suited for line drawings.

GIF (Graphics Interchange Format): This is a non-lossy, bitmap based file format widely used in web browsers and other applications. It was defined by Compuserve Inc. in 1987 and 1989. The image can have up to 256 colors and 16'000 x 16'000 pixels. For compression the LZW (Lempel-Ziv-Welch) algorithm is used, patented by Unisys.

PNG (Portable Network Graphics): This is an extensible file format for the lossless, portable, well-compressed storage of raster images, supported by a large number of applications. PNG is the most modern format with excellent compression. PNG provides a patent-free replacement for GIF and can also replace many common uses of TIFF. Indexed-color, grayscale, and truecolor images are supported, plus an optional alpha channel. Sample depths range from 1 to 16 bits.

TIFF (Tag Image File Format): This is a non-lossy, bitmap based file format supported by a large number of applications. TIFF describes image data that typically comes from scanners, frame grabbers, and paint- and photo-retouching programs.

Appendix C: Apple Script Cross Reference

The following table is a cross reference (incomplete) between the functions and procedures of pro Fit's built-in programming language and equivalents available under Apple Script.

pro Fit programming language	Apple Script equivalent
–	do script
–	evaluate
–	get data
–	quit
–	set data
AddParameterSet	add parameter set
AttachProgram	attach
BinData	bin data
CallProgram	run program
Capture	capture
Clear	clear
CloseWindow	close
Compile	compile, do script
Copy	copy
Cut	cut
DataExportOptions	data export options
DataImportOptions	data import options
DataTransform	transform
data[i,j]	*use the* cell *elements of the data window, e.g.* get value of cell 1 of column 2
DeleteFunction	delete
DeleteFunction	delete function
DeleteParameterSet	delete parameter set
DeleteProgram	delete
DeleteProgram	delete program
DeleteShape	delete shape

DeleteTag	*use* delete tag
Extrema	find extrema of
FFT	FFT
Fit	fit
GetColumnProperty	*use the properties of the* column *class, e.g.* get format of column
GetDataWindowProperty	*use the properties of the classes* window *or* table, *e.g.* get nrRows of window "MyData"
GetFunctionProperty	*use the properties of the class* function, *e.g.* get nrParams of function "Sin"
GetGlobalData	*use* get global data
GetOption	*use the properties of the pro Fit application class, e.g.* get decimals
GetParameterProperty	*use the properties of the class* parameter, *e.g.* get name of parameter 1
GetProgramProperty	*use the properties of the class* program, *e.g.* get idleCallTime of program "Prog"
GetResult	*use the* results *property and the* calculation results *class, e.g.* get chiSquared of results
GetShapeProperty	*use the properties of the class* shape, *e.g.* get value of shape "checkbox 1"
GetTag	*use* get value of tag
GetWindowProperty	*use the properties of the classes* window, drawingWindow *or* table, e.g. get name of front window
Integrate	integrate
InverseFFT	inverse FFT
LoadParameterSet	load parameter set
NewDataWindow	make table
NewDrawingWindow	make drawingWindow
NewFunctionWindow	make textWindow
NewShape	make shape
OpenData	*use* open file "..." as data window
OpenFile	open
OpenText	*use* open file "..." as text window

`Optimize`	optimize
`PageSetup`	page setup
`Paste`	paste
`PlotData`	plot data
`PlotFunction`	plot
`Print`	print
`ReduceData`	reduce data
`Roots`	find roots of
`SaveParameterSet`	save parameter set
`SaveWindow`	save
`SelectAll`	select all
`SelectCell`	*use select, e.g.* select cell x of column 1
`SelectColumn`	*use select, e.g.* select column 3
`SelectFunction`	*use select, e.g.* select function "Sin"
`SelectRow`	*use select, e.g.* select row 4
`SelectWindow`	*use select, e.g.* select window "Results"
`SetColumnProperties`	*use the properties of the* column *class, e.g.* set name of column 3 to "data"
`SetDataWindowProperties`	*use the properties of the classes* window *or* table, *e.g.* set nrCols of window "MyData" to 50
`SetFitParamRange`	set fit range of parameter
`SetFunctionProperties`	*use the properties of the class* function, *e.g.* set shown of function "Sin" to true
`SetGlobalData`	*use* set global data
`SetLegendProperties`	set legend properties
`SetOptions`	*use the properties of the pro Fit application class, e.g.* set default column type to real
`SetProgramProperties`	*use the properties of the class* program, *e.g.* set idleCallTime of program "Prog" to 131203
`SetShapeProperties`	*use the properties of the class* shape, *e.g.* set xSize of shape "rect" to 231
`SetTag`	*use* set value of tag
`SetWindowProperties`	*use the properties of the classes* window, drawingWindow *or* table, e.g. set font of window "Func" to "Helvetica"

Sort	sort
Statistics	calculate statistics -- use pro Fit's property "results" for retrieving the results, e.g.
	get statMean of results
Tabulate	tabulate
TabulateExtrema	tabulate extrema of
TabulateIntegral	tabulate integral of
TabulateRoots	tabulate roots of
Transpose	transpose
Undo	undo
Write	write
WriteLn	write line

Index

normal style, 105
nrRows, 141
number of digits, 223
numerical accuracy, 242
Nyquist critical frequency, 42
`ok`, 165
operators, 160
optimize, 50
optional parameter lists, 158
output, 176
outputs
  default values, 162
page setup, 217
parameter defaults, 164
parameter limits, 48, 115, 119
parameter limits, 127
parameters, 119
  defaults, 140, 162
  mode, 140
  poor definition, 134
  redundancy, 133
parameters window, 161
params -->>, 125
partial derivatives, 114, 115, 122, 166
partial derivatives, 203, 204
Pascal, 187
paste, 79
pb, 201
pdf, 220
PDF files, 75
PICT, 219
  files, 74
  options, 74, 77, 219
PICT style, 220
picture settings, 220
pictures, 74, 79, 219
plot
  data, 86, 89
  function, 82
  tabulating, 100
plots, 98
plotting
  preferences, 224
plotting, 79
plug-ins, 11, 189, 191, 199
  loading automatically, 190
PNG files, 77

`pNumber`, 165
pointers, 188
points, 49, 69, 98, 240
polygons, 67, 239
population, 16
power operator, 141, 160
precedence, 160
precedence, 161
preferences, 104, 222
  Drawing, 223
  Extensions, 231
  File export, 230
  Functions, 228
  General, 222
  Interface, 227, 229
  PICT Options, 219
  Plotting, 224
  Prefs file, 232
  Preview, 226
  printing, 217
preferences file, 191
  loading and saving, 232
preferences files, 191
preview, 54
  drag tool, 57
  fitting tool, 57
  markers, 57
  preferences, 226
  zoom tool, 57
preview window, 123, 125
preview window, 54
printing, 217
  at printer's resolution, 217
pro Fit plug-ins, 190
pro Fit plug-ins folder, 190
pro Fit preferences, 222
probability axes, 82
procedure, 154, 162
  Derivatives, 166
  First, 167
  Initialize, 166
  Last, 168
program
  attaching to windows, 191
  definition, 136, 149, 152, 236
  linking, 199
program plug-in, 190