FIT2.EXE User's Guide

Michael Courtney

February 11, 1994

Contents

1 Installation

Unzip the zip file in a suitable directory. Move *fit2.exe* somewhere in your path. If you want the online help to work, set the environment variable *fithelp* to point to the file *fit.hlp*. For example:

```
set fithelp=d:\docs\fit.hlp
```

Under Unix, you should use setenv, something like:

```
setenv FITHELP /u/docs/fit.hlp
```

Adding the line to your *config.sys* (under OS/2) or *autoexec.bat* (under DOS) or .cshrc would keep you from having to type it in every time you reboot or login. The gnuplot.exe program should be somewhere in your path. The fit program requires you have *gplt33b2.zip* or later under OS/2. Earlier versions work under Unix. This file is available from ftp.cdrom.com and includes complete documentation. As of 2/11/94 the most recent version of gnuplot is 3.5.

For use on platforms other than OS/2 2.x or DOS, you will have to compile. Have a look at the makefile and make it suitable for your compiler. You might want to make the executable *fit2* instead of *fit2.exe*. You should also use the compiler option -DUNIX (instead of -DOS2), or whatever option will define UNIX for your compiler. I recommend using the GCC compiler for the most painless compilation on any UNIX platform, but other compilers have been shown to work as well. You might get some warnings.

Under OS/2 2.x, you should move the *.DLL files to a directory in your LIBPATH.

Under DOS, you should use the DJGPP port of GCC. Borland C++ will not work. To run in a DOS box under Windows, you must use version 1.10 or higher of DJGPP. Specify the -DDOS option when you compile under DOS. Plotting is disabled in the DOS version. You will have to write the fit to a temporary file, and plot with the plotting package of your choice. DJGPP is not the most usable DOS compiler, but it is free, has a 32 bit DOS extender, and is good for porting UNIX programs. Consult the DJGPP documentation and the *.msdos.programmer newsgroups for more info on DJGPP. You must have the file *go32.exe* in your path for the DOS version to work.

-DUNIX affects the command used to open gnuplot. Gnuplot writes some error messages to stderr and some of these make it to the screen. If -DUNIX is specified, stderr is redirected to the file gnuout. If you want to see the error messages, do not use -DUNIX. Additionally, some gnuplot terminal types require stderr to go to the screen. If you have one of these, do not specify -DUNIX; you will have to live with the gnuplot error messages. This is the case for *tek40xx* and similar terminal types. If the -DUNIX option is not specified, temporary files will be written to the current directory rather than */tmp*.

2 General

fit2 is a non-linear least-squares fitting program. It uses the Levenberg-Marquardt method to attempt to minimize the least-squares error between a data file and a model function. See the books, **Data Reduction and Error Analysis in the Physical Sciences** by P. R. Bevington and **Numerical Recipes in C** by W. H. Press et. al. for more information on this method.

Due to the generality of the fitting function allowed, and the possible weirdness of the error function in parameter space, convergance to the true minimum is not guaranteed. Therefore, it is important to have good initial guesses for the parameters. It is a good idea to graph your data and the fitting function with your initial guesses, and tweak the initial guesses until the fitting function is close to the data. Most functions with three or four parameters will converge quickly if the initial guesses are in the right ballpark. If you have more than four parameters, you may have to tell the program to vary only four at a time (the **sp** command). When the parameters you are varying affect a subset of the data much more than the rest, you should window the data accordingly (the **wi** command). If these tricks do not work, check and make sure that the function is computing the proper value of the function and derivatives with respect to the parameters. You have to check your source for this. A sign error is disasterous. It will cause the guess for the parameter to be further from the correct value than the current value.

The Levenberg-Marquardt method will work for functions which are linear in the parameters. It is not as fast as linear regression, and it is not guaranteed to converge to an absolute minimum. However, it will do in a pinch and it is actually a little more robust if your basis functions are nearly linearly dependent. Of course, if your basis functions are nearly linearly independent then your parameters will be strongly correlated, and you will have to sort what is meaningful and what is not.

This program was written by Michael Courtney. I consider this a beta version of the program. I retain all rights to the program. You may use the program and alter the source to fit your needs. You may not distribute altered source or executable unless you obtain my permission. You may port this program to other platforms if you wish. It should compile and run almost unaltered on the NeXT platform and most Unix platforms. You may not sell the program or source or receive any money for its distribition. The program is provided *as is* and is without warranty of guarantee of any kind. The author assumes no responsibility of any kind for anything the program does.

Please report bugs and suggestions to Michael Courtney at michael@amo.mit.edu. I will consider bugs to have a higher priority than suggestions, although suggestions are welcome.

I am considering the following future enhancements to the program:

- Multiple data sets – the ability to simultaneously fit multiple data sets to different functions which share common parameters. Currently, you can effectively accomplish this, see Multiple Data Sets.
- Ability to handle binary files.
- Command line history.

Please let me know if any of these enhancements are of interest to you.

3 Improvements

This is the 1.5 beta version of the program.

The improvements over the 1.4 beta are the following:

- Can now take commands from a file. See the **do** command.
- Can now use the Cset++ compiler under OS/2 2.x.
- Minor bug fixes.
- The **noset** command.

The improvements over the 1.3 beta are the following:

- New plotting routine under OS/2 allows viewing of a plot of every attempted fit. See the **fp** command.
- Minor bug fixes.
- A simple function for estimating errors in the parameters has been added.

The improvements over the 1.2 beta are the following:

- Linear Least-Squares Fitting
- Minor bug fixes.
- Can change one parameter at a time. See the **cp** command.

4 Files

fit2 opens and stores data in several files. *fit.tmp* is used if the function is plotted to a file. *fit.dat* is used as a default filename if none is specified with the **wf** command. The **wp** command stores the parameters in the file *a.dat*. In addition, the text output of gnuplot which is sent to stderr is written in the file *gnuout*. On non-unix machines, *fit2* does not erase any of these files, unless it overwites them. On UNIX machines *fit.tmp* and *gnuout* are written in the */tmp* . They are erased when the program exits, unless debugging is on. If the debug flag is non-zero, lamda is written to a file called *lamda.sts* on every successful iteration. lamda controls how much the paramerers are changed in attempting to reduce χ^2, and changing how lamda is chosen would be one way to improve effeciency of the L-M algorithm.

5 Commands

The following can be executed at the `fit2>` prompt:

help

quit

fn function

ip initialize parameters

cp change one parameter

sp select parameters (to vary)

gd get data

fi fit

pp print parameters

wp write parameters (to a file)

wf write fit (data to a file)

rp read parameters (from a file)

wt weight (for fitting)

co covariant matrix

order order (of data colums in file)

ad allocate data array

lf lists functions available

gr graphing flag

plot plot data and fit

load load a gnuplot command file

set set something in gnuplot

wi windows data

ve changes verbosity

noset changes how *fit2* sends set commands to gnuplot.

pr plot residual errors

sh shows current settings

md makes data

de debug

ru run a program

gn sends a command to gnuplot

li linear least squares fit

pa pauses for a number of seconds

fp turns the use of fitplot.exe on and off

er estimates errors in the parameters

do executes commands in a file

5.1 fn

Usage: `fn name [#]`

fn chooses a function for fitting. If the function has a variable number of parameters, the number of parameters can be entered after the function name. **Example:** `fn gauss` chooses to fit to a gaussian.

5.2 ip

Usage: `ip a0 a1 a2 ...`

ip initializes the fitting parameters. If the number of parameters changes, the parameters are initialized to 5 when the **fn** command is executed. **ip** can be used at any time to change the values of the parameters. If you want to leave some parameters unchanged,enter a * in place of the number. You must enter either a number or a * for all of the parameters.

Example: `ip 13.7 * 12` changes a0 to 13.7, leaves a1 the same, and changes a2 to 12 for a three parameter fit.

5.3 cp

Usage: `cp # #`

cp changes the parameter specified by the first number to the value specified by the second number. The first number must be in the range of parameters.

Example: `cp 3 3.141` changes a3 to 3.141 and leaves all the other parameters unchanged.

5.4 sp

Usage: `sp # # #`

sp selects which parameters to vary. By default, all parameters are selected to be varied when the **fn** command is executed. If you only wanted to vary parameters a0 and a2 of a three parameter fit, you would issue the command:

Example: `sp 0 2`

5.5 gd

Usage: `gd filename`

gd geta the data from a file. Data should be in an ascii file with between two and 20 columns. Actually, you can have as many columns as you want in the file, but the program will only read up to 20. For fitting functions of more than one independent variable, select a function with the number of independent variables that you want to fit to before reading in data. This will allow the program to default to interpreting columns 1 through n (where n is the number of independent variables of the current function) as representing the independent variables in order. By default, the (n+1)th column is interpreted to be y(x0 ... xn) and the (n+2)th column is interpreted as the experimental error in the measured y. If you wish to assign the columns in your data file differently, or if you read in the data before choosing a function (for more than 1 independent variable), you will have to use the **order** command.

By default, if no function has been selected, or if a function of one independent variable has been selected, the first column is x, the second is y, and if there is a third, it is assumed to be the error in y. If you wish to assign different columns to be x, y, and δy, use the **order** command. By default, the program can handle a file with five columns and 1024 rows. The **ad** command can be used to allocate the data array differently and handle up to 23 columns and as many rows as memory allows.

Lines in a data file beginning with the `#` character are taken to be comments.

5.6 fi

Usage: `fi iterations`

fi fits the data using the non-linear fitting algorithm. Iterations is the number of iterations between plots. If no number it given, it defaults the last number used. If none was ever entered, it defaults to 20. After this number of iterations, the data and fit are plotted, if graphing if selected. (It is by default. See the **gr** command.) You are then given the option to quit iterating (q enter), continue iterating and turn on graphing (g enter), continue iterating and turn off graphing (n enter), or continue with the same graphing status.

If the **fi** command is executed from a command file, the fit stops after the selected number of iterations and does not prompt for user input. This is to facilitate batch processing.

There is no way in general to be sure χ^2 is at an absolute minimum. It might just be at a local minimum. Starting the fit with several sets of values of initial parameters and having every fit that converges find the same set of final values and same χ^2 gives some confidence. Looking at the graph at least tells us we're not too far away.

It is possible to estimate the errors in the parameters once χ^2 is minimized. Rather than build this into the program for people to use with more confidence than warranted, I merely make all the information you need available. You should look at the references and figure it out. This way you will have an idea of how unreliable error estimates on the parameters can be.

5.7 li

Usage: `li`

li fits the data using a linear fitting algorithm. No iteration is required since the best-fit parameters are uniquely determined by the data and basis functions. The linear fitting algorithm used is the simplest possible. A matrix equation is set up and solved by Gauss-Jordan elimination. A singular matrix will cause a failure of the method and probably indicates a linearly dependent basis set. A near-singular matrix will cause inaccuracies in parameter values.

li will usually work very well for less than 6 or so basis functions and reasonably well for up to about 10 basis functions. If you have problems with singular matrices, doubt the accuracy of your parameters, or want to incorporate errors in the independent parameters, I suggest that you use **li** to obtain initial guesses and then use **fi** for to find the final parameter values. **fi** will usually converge quickly if you start with initial guesses obtained from **li**.

You should be aware that fitting to a subset of parameters with the **sp** command has a different effect with **li** than it does with **fi**. With **fi**, the selected parameters are varied while the unselected parameters are held fixed. With **li**, the selected parameters really define which basis functions are incorporated in the fit. These are the only parameters found, the other parameters are set to zero. This may be inconvenient in certain situations, but it reflects a basic difference between linear and non-linear fitting.

5.8 pp

Usage: `pp`

pp prints the current value of the parameters to the screen.

5.9 wp

Usage: `wp filename mode`

wp writes the parameters to a file. If no filename is given, parameters are written to the file *a.dat*. Mode should be *a* (for append) or *w* (for overwrite). The default mode is *w*.

Examples:

`wp` writes the parameters to the file *a.dat*, overwriting the file if it exists.

`wp params.dat a` writes the parameters to the file params.dat, appending then on to the end of the file if it already exists.

5.10 wf

Usage: `wf filename`

wf writes the fitted data to a file, overwriting the file if it exists. If no filename is given, data is written to the file *fit.dat*. The fitted data is the value of the fitting function evaluated at the value of x of the current data set using the current parameters.

5.11 er

Usage: `er factor`

er provides a simple way to estimate the errors in the parameters. The algorithm is very simple, and the answer it gives is intuitively satisfying, but not statistically rigorous. The algorithm varies one parameter at a time and finds two values of the parameter where chisqr is increased by the given factor. An increase of $10 - 20\%$ in chisqr should give a reasonable estimate of the errors in the parameter, so I suggest a factor of 1.1 or 1.2. For factors larger than 2.0, the algorithm may not converge, and you answers would not make sense as error estimates on the parameters.

5.12 md

Usage: `md filenaeme xmin0 xmax0 xstep0 xmin1 xmax1 xstep1`

md makes a trial data set. It's primary use is in testing new fitting functions. At this time, it is only implemented for functions of one or two independent variables. The trial data set is created using the current parameter values and the current fitting function.

Examples:

```
fit2> fn gauss        /* defines fitting function as gauss */
fit2> ip 5 1 1        /* initialized parameters */
fit2> md test 0 10 1  /* makes data for 0 <= x0 <= 10*/
fit2> gd test         /* reads in data */
fit2> ip 5.1 2 1      /* changes parameters */
fit2> fi              /* test to see if fit finds real parameters */
fit2> fn xyquad       /* defines fitting function as xyquad */
fit2> ip 5 5 5 5 5 5  /* initialized parameters */
fit2> md test 0 10 1 0 10 1
                      /* makes data for 0 <= x0 <=10 and 0 <= x1 <= 10*/
fit2> gd test         /* reads in data */
fit2> ip 1 1 1 1 1 1  /* changes parameters */
fit2> fi              /* test to see if fit finds real parameters */
```

5.13 rp

Usage: `rp filename`

rp reads the parameters from a file. If no file is given, it attempts to read the parameters from the file *a.dat*.

5.14 de

Usage: `de flag`

de changes the debugging status of the program. Flag is 0 for no debugging, 1 for light debugging, and 2 for heavy debugging. Only certain sections of code have been set up for this. Other sections will be set up as I need to debug them. This option is intended mainly for use of someone altering the program. If the debugging level is 1, all of the commands sent to gnuplot are printed, along with other information.

5.15 sh

sh shows the value of current settings.

5.16 wt

Usage: `wt [none stat inst other]`

The fitting function tries to minimize χ^2, where $\chi^2 = \sum\left((y_i - f(x_i))/\sigma_i\right)^2$. That is, χ^2 is the sum of the squared error of the difference between the data and the fitting function, except that each term in the sum is weighted by σ_i^2. The **wt** command chooses the type of weighting. It chooses what to use for σ. Errors in the independent parameters can also be used for non-linear fitting. See the order command for details. The following weighting options are available:

none..........no weighting, all σ's are equal to one.
statistical...$\sigma_i = \sqrt{y_i}$
instrumental..$\sigma_i = \delta y_i$, as in data file
other.........$\sigma_i = y_i$
Example: `wt s` selects statistical weighting.

5.17 co

Usage: `co filename`

co prints the covariant matrix. It will overwrite the file if a file of the same name exists. If no filename is given, the covariant matrix is written to the screen. The covariant matrix can be used to estimate the error of the fit parameters.

5.18 order

Usage: `order x0col ... xncol ycol` σ_y`col` σ_{x0}`col ...` σ_{xn}`col`

order assigns significance to the columns in the internal data matrix. When a data file is read with the **gd** command, the first column in the file is stored in the data[0]. (The 0th column of an 2-D array). The second column is stored in data[1], etc. By default, these columns are interpteted in a certain way (See the **gd** command). The **order** command is used to change the default behavior.

Errors in the independent variables can be used in the fit by using techniques of error propagation to calclate a corresponding error in y. That is:

$$\sigma_y = \sqrt{\sigma_y^2 + \sigma_x^2 \frac{dy}{dx}^2}.$$

Each data point has its own value of σ_x and σ_y. Each σ_x must be in a column in the data file. The weighting for σ_y should be statistical, instrumental or other if σ_x is considered in the fit. You should not select no weighting if you want to use σ_x. If the **order** command selects a column to be σ_x, then this information is used for the fit. If you wish to not use this information, assign $\sigma_x$col to be -1 (the default value). If the fitting algorithm has a problem converging when using σ_x, try the fit without using σ_x. Then take those parameters as the initial parameters for the fit considering σ_x. Note that errors in the independent variables are not considered during linear least squares fitting.

Examples for one independent variable:

`order 2 1 0` x0 is the third column in the file, y is the second column, σ_y is the first column.

`order 1 4` x0 is the second column, y is the fifth column, σ_y is not specified.

`order 0 1 2 3` x0 is the first column in the file, y is the second column, σ_y is the third column, σ_{x0} is the fourth column and is used in the fit.

`order 0 1 2 -1` x0 is the first column in the file, y is the second column, σ_y is the third column, σ_{x0} is not used.

Examples for two independent variables:

`order 2 1 0 3` x0 is the third column in the file, x1 is the second column, y is the first column, σ_y is the third column.

`order 1 4 2` x0 is the second column, x1 is the fifth column, y is the third column, σ_y is not specified.

`order 0 1 2 3 4 5` x0 is the first column in the file, x1 is the second column, y is the third column, σ_y is the fourth, σ_{x0} is in the fifth column and is used in the fit, σ_{x1} is in the sixth column and is used in the fit.

`order 0 1 2 3 -1 -1` x0 is the first column in the file, x1 is the second column, y is the third column, σ_y is the fourth, σ_{x0} and σ_{x1} are not used in the fit.

5.19 ad

Usage: `ad columns rows`

ad re-allocates the internal data matrix. It is 8 columns and 1024 rows by default. You can have up to 23 columns and as many rows as memory allows. Three rows are used internally, and up to 20 rows can be used for data from your input file. The **ad** command erases any data currently in memory.

Example: `ad 11 4096`

5.20 lf

Usage: `lf`

lf lists the functions available for fitting.

5.21 gr

Usage: `gr flag`

The **gr** command turns the graphing on and off. `gr 0` turns graphing off. `gr 1` turns graphing on. Graphing is on by default.

5.22 fp

Usage: `fp flag`

The **fp** command turns the use of the *fitplot.exe* program on and off. The *fitplot.exe* program is a fast plotting program which is used to plot the data and attempted fit every iteration. This is useful if you want to see what the Levenberg-Marquardt algorithm is doing. The use of fitplot.exe is off by default, although if there are any command line arguments to *fit2.exe*, the use of *fitplot.exe* is turned on.

5.23 plot

Usage: `plot`

plot plots the data and fitting function with the current parameters.

5.24 load

Usage: `load filename`

load loads and executes a gnuplot command file.

5.25 set

Usage: `set gnuplot stuff`

The entire **set** command is sent to gnuplot.

5.26 wi

Usage: `wi [one or more numbers]`

wi selects data windowing. It is used if you want to fit the data in a certain range of x values. If **wi** has one argument, it turns windowing on and off. `wi 1` turns windowing on. `wi 0` turns windowing off. If **wi** has more than two arguments, windowing is turned on, and the arguments are the minimum and maximum x values used in fitting. In other words, χ^2 is computed only for $xmin \leq x \leq xmax$.

Examples:

`wi 30 100` fit considers only x values between 30 and 100.

`wi 30 100 2 10` fit considers only x0 values between 30 and 100 and x1 values between 2 and 10.

5.27 ve

Usage: `ve flag`

If flag = 0, the **fi** command gives little output. If flag = 1, the **fi** command only gives the paramters every iteration. If flag = 2, the output is very verbose.

5.28 noset

Usage: `noset flag`

If flag = 0, then fit2 issues the set command to gnuplot often for setting ranges, labels, titles, etc. If flag = 1, fit2 only sends gnuplot a set command when absolutely necessary for performing the requested operation.

5.29 pr

Usage: `pr flag`

pr plots residual errors. If there is no flag or flag = 1, yfit vs. y is plotted. If flag = 2, (y-yfit) vs. x is plotted.

5.30 run or !

Usage: `run command line`

ru sends a command line to the default command processor. It is used to run a program from within *fit2*.

Example: `run dir *.dat`

5.31 gn

Usage: `gn gnuplot command`

gn sends a command to the gnuplot command processor. It is used to execute gnuplot commands from within *fit2*. You need to send a correct gnuplot command. You will get no error message if you do not.

5.32 pa

Usage: `pa seconds`

If $seconds < 1$, the program is paused until enter is hit.

5.33 do

Usage: `do file`

do executes commands from a command file. Lines beginning with **#** are comments. Blank lines are skipped. All other lines are executed. You cannot call one command file from another under OS/2. It might work under Unix.

6 functions

There are several built in functions. Other functions can be added by adding them to the file funclib2.c and recompiling. See that file for details. The built in functions are: gauss, gaussc, ngauss, lorenz, 2lorenz, line, poly, nexp, and xyquad.

6.1 gauss

The function gauss is a gaussian of the form:

$$f(x) = a_2 e^{-(\frac{x-a_0}{a_1})^2}.$$

Fitting to this function converges easily if your initial guesses are even close. For gauss and most peak-type functions, your initial guesses will probably converge if your function peaks are a little shorter and wider than you data peaks.

6.2 gaussc

The function gauss is a gaussian plus a constant.

$$f(x) = a_2 e^{-(\frac{x-a_0}{a_1})^2} + a_3.$$

Fitting to this function converges easily.

6.3 ngauss

This function is a sum of gaussians plus a constant. It uses a variable number of parameters. The number you want must be chosen when you select the function. The function has the form:

$$f(x) = a_{i+2} e^{-(\frac{x-a_i}{a_{i+1}})^2} + a_{n-1}.$$

You must choose $3n + 1$ parameters, where n is the number of gaussians you want. This function converges slowly for more than 2 gaussians, but it usually does converge. If the peaks in your data are separated by several widths, you should window the data and fit the peaks separately first, and then do a final fit on the whole data set.

6.4 lorenz

This is a lorenzian function of the form:

$$f(x) = \frac{a_2 a_1}{4(x-a_0)^2 + a_1}.$$

I chose not to use the usual form, because I thought making the form simple in regards to the parameters might make for more robust fitting. You have to think to eyeball the initial paremeters though. If your data is a lorenzian, it will converge without much trouble. If your data is not really a lorenzian you may have trouble. The gaussian is a better choice to fit peaks that are not well characterized by a particular function.

6.5 2lorenz

This is the sum of two lorenzians and it has the form:

$$f(x) = \frac{a_2 a_1}{4(x-a_0)^2 + a_1} + \frac{a_5 a_4}{4(x-a_3)^2 + a_4}.$$

It sometimes takes some effort to get a fit to this function to converge.

6.6 line

Piece of cake:

$$f(x) = a_0 + a_1 x.$$

6.7 poly

This function is a polynomial of the form:

$$f(x) = \sum a_i x^i.$$

Decent initial guesses and fourth order or less converge easily and quickly. Higher orders might take some work. Using **li** to obtain initial guesses and **fi** for final parameter values is best for polynomials higher than fourth order.

6.8 nexp

This function is the sum of decaying exponentials. It has the form:

$$f(x) = \sum \Theta(a_i) a_{i+2} e^{\frac{x-a_i}{a_{i+1}}} + a_{n-1},$$

where $\Theta(a_i) = 0$ if $x < a$, and $\Theta(a_i) = 1$ if $x \geq a$. You need to tell the program how many parameters you want. 4 parameters gives you one exponential, 7 gives you two, etc. If you know the offset, you should initialize properly and not vary that parameter. For example, most decaying exponentials are triggered by some event. If you know the time of the event, don't vary that parameter.

6.9 xyquad

This is a two dimensional quadratic of the form:

$$f(u, v) = a_0 + a_1 u + a_2 v + a_3 u^2 + a_4 v^2 + a_5 uv.$$

u and v are used instead of x and y or x0 and x1, because gnuplot needs to plot the data parametrically. The linear fitting algorithm works pretty well, but I recommend the result be checked with a few iterations of the L-M algorithm, for example, issue the command, `fi 5` after the **li** command.

6.10 sincos

This function is a sum of sines and cosines of the form:

$$f(x) = a_0 + a_i \sin a_{i+1} x + a_{i+2} \cos a_{i+3} x.$$

You need good initial guess to get this fitting function to converge quickly, if at all. You will do well to get the frequencies close. If the zeros of your fitting function with your initial parameters are close to the zeroes of your data, you have a good chance of converging. For special cases like Fourier series, you probably want a specialized function. You should take a Fourier transform if that is what you really want.

6.11 conic

conic does not work very well and I do not reccomend its use.

7 Multiple Data Sets

Let's say that you have several different data sets that you want to fit to functions which share some common parameters. You can simulate the functionality of multiple data sets.

Put all of your data in the same file , with the same columns being used for x0, x1 . . . xn, y, any sigma's etc. Add an extra row to the file containing a number to uniquely identify the data set. Define your function for n+1 independent variables, and assign the extra row to be x(n+1). The (n+1)th independent variable will be passed to your fitting function. Use it to decide which data set you are using and return the appropriate function value and derivatives.

Sure, it's a kludge, but it's better than nothing for now.

8 gnuplot

Gnuplot is is an interactive plotting program. To use the graphing capability of the fit program, you must have the gnuplot program correctly installed in your system and it must be in a directory in the current path.