# An Introduction to Gnuplot for Chemists

By Nicholas Fitzkee
Mississippi State University
Updated May 19, 2021

*Introduction*

Many biophysical problems involve fitting a model (as described by some mathematical function) to experimental data. Typically, these models have one or more parameters that must be optimized in order to obtain the best fit. Then, once the optimal parameters have been found, the parameters can be interpreted and compared. An example of this might be fitting enzyme kinetics data: by fitting the Michaelis-Menten model to kinetic data, one can find the best values of $v_{max}$ and $K_M$ (the parameters for that model).

Fitting is a complex subject, and sophisticated models often require the assistance of a computer to perform the fitting. As a result, several programs have been written to perform model fitting. Many of these programs are highly sophisticated themselves: Matlab is a prime example of a program that can do model fitting as well as many other computations. Others are less sophisticated, but still powerful: Microsoft Excel includes a simple fitting routine, even though it isn't statistically rigorous.

Our challenge is to find a program that is easy to use, free, and yet powerful enough to produce publication-quality fit results. In the past, Michael Johnson's program NONLIN has been used for this purpose, but it is no longer maintained. Instead, we will use Gnuplot (pronounced "NEW-plot") to do our fitting. Gnuplot is a free program that was originally designed to create publication quality graphs for scientists. However, it also has some nice features for model fitting.

Gnuplot contains a wealth of features, and you may want to consider using it for your research (many people do this). One of the best online tutorials for Gnuplot is available through IBM, and there are several others out there. We will focus on fitting in Gnuplot, but if you'd like to know more about the program, you can check out these websites:

- http://fitzkee.chemistry.msstate.edu/sites/default/files/ch8613/ibm-gnuplot.pdf
  This is IBM's tutorial for developers, written by Nishanth Sastry. It's well written, and it's great PR for IBM. At least it was, until it was removed from the IBM website.

- http://www.duke.edu/~hpgavin/gnuplot.html
  This is Dr. Henri Gavin's Tutorial at Duke. It contains some information on fitting as well as other basic functionality.

You can find many more introductory tutorials for Gnuplot on the web, and the Gnuplot home page also hosts plenty of documentation itself.
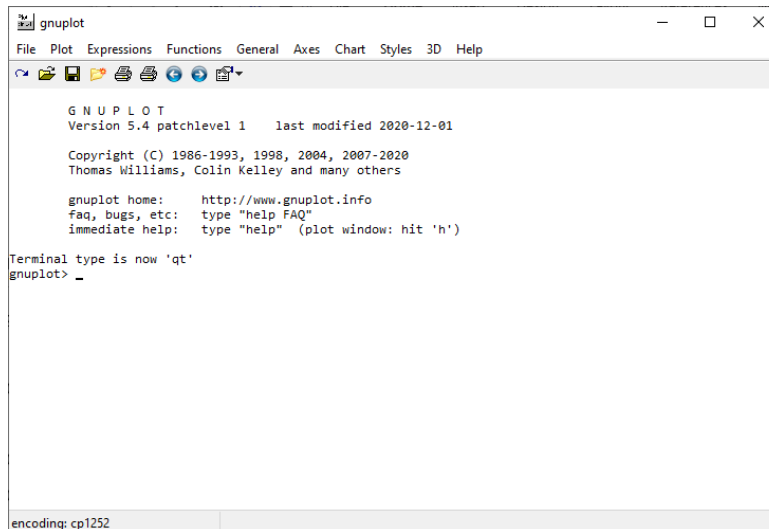
*Installing Gnuplot*

If you have an account on a linux system, you're in luck: there's a good chance that Gnuplot is already installed. Simply access a terminal window and type `gnuplot`. If everything is installed, you'll see some information on the terminal followed by a prompt (see below). Otherwise, you can always use the gnuplot installed on the course linux server.

That said, I expect many of you will not want to bother with a UNIX environment. Fortunately, Gnuplot binaries are available for Windows systems, and you can download them from the website: http://www.gnuplot.info/. The current version can be obtained using the Download link at the top of the webpage and following the links from there. You will need to download the non-X11 version of the software (currently, this is either gp506-win32-mingw.exe or gp506-win64-mingw.exe depending on whether you have a 32- or 64-bit system). Unfortunately, Gnuplot is not available for Mac OSX systems.

To install Gnuplot, you must download an installer file from http://www.gnuplot.info/. The download page lists both PC and Mac versions. Mac versions are generally listed directly at the bottom of the download page. PC versions can be found by clicking on the "Primary SourceForge Download site." Look for the "latest version link; the file name (for 64-bit) will be something like `gp523-win64-mingw.exe`. The installer works like many other Windows installers, and the default installation options will be sufficient for 99% of users. Gnuplot will create a program icon on your start screen (or start menu), and you can optionally place an icon on your desktop as well. The easiest way to start Gnuplot is to click the start button and begin typing "Gnuplot." Eventually, Windows will find the program and you can click on it to run the software. Should you want to upgrade or remove Gnuplot, it can be uninstalled easily through the "Add/Remove Programs" widget in the control panel.

Once Gnuplot has been opened, you will see a window like this:



On the Linux version, you won't see the buttons at the top, but the overall screen should be similar.

*Your First Plot*

For simplicity's sake, let's start by examining how you can plot functions in Gnuplot.  At the prompt (`gnuplot>`), type the following:

```
plot [-10:10] x**3
```

When you press return, you should see a window that plots the function $f(x) = x^3$ on the range [-10, 10].  If this is your first time running Gnuplot, it may take a while for this window to finish, but eventually you should see something like this:
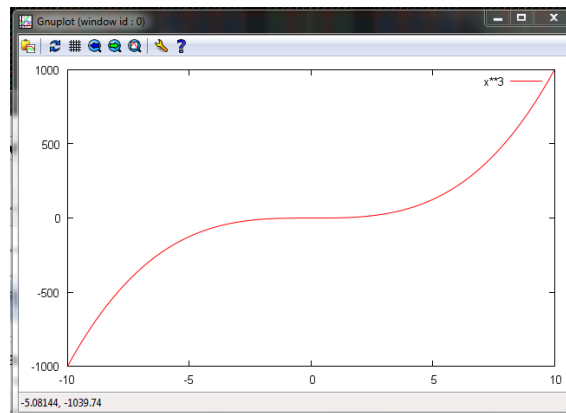


Thus, the command above creates a new plot.  The syntax of the plot statement is simple.  The first parameter, [-10:10], tells the program use a range from -10 to 10.  If you leave out this parameter (e.g., `plot x**3`), Gnuplot will try to figure out the best range to use itself.

The second parameter tells Gnuplot what function to plot – note that the double asterisk (**) means "raised to the power," so `x**3` means $x^3$.  You can use multiple terms in your function plotting.  If you wanted to plot a more complex polynomial, you could write:

```
plot [-10:10] 10*(x**2) - 4*x + 3
```

Which would plot $f(x) = 10x^2 - 4x + 3$.  The parentheses around the `x**2` aren't strictly required, since Gnuplot will perform exponential operations before multiplications, but it makes the statement more clear.  For those of you who have already started your programming assignment, you will find the syntax of these expressions to be eerily familiar: Gnuplot uses much the same syntax as C or FORTRAN, and it shares many of the same functions as well.  Some examples are found in Table 1.

The error function should appear familiar to you from your reading.  Specifically, you should be able to show that $\mathrm{erf}\left(\frac{n}{\sqrt{2}}\right)$ is the probability for an observation to appear within $\pm n$ standard deviations from the mean.

**Table 1.** Common functions used in Gnuplot. Note that trigonometric functions are in radians (not degrees!)

| Function | Return Value |
|----------|--------------|
| `abs(x)` | $\lvert x \rvert$ |
| `acos(x)` | $\cos^{-1}(x)$ |
| `asin(x)` | $\sin^{-1}(x)$ |
| `atan(x)` | $\tan^{-1}(x)$ |
| `cos(x)` | $\cos(x)$ |
| `erf(x)` | The error function, i.e. $f(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ |
| `log(x)` | $\ln(x)$ |
| `log10(x)` | $\log(x)$ |
| `sin(x)` | $\sin(x)$ |
| `sqrt(x)` | $\sqrt{x}$ |
| `tan(x)` | $\tan(x)$ |

One of the reasons that Gnuplot is powerful is that it's easy to examine functions without setting up Excel spreadsheets – you simply need to type the functional form into the prompt, and examine how the curve behaves. You can even use simple variables to make it easy to change your function. Enter the following commands:

```
a = 1
b = 4
plot [-5:5] a*x+b
```

These commands plot a line with a slope of 1 and an intercept of 4. But suppose you wanted to quickly examine the effect if the slope were -1? You could simply type:

```
a = -1
replot
```

The plot is re-displayed, this time with the updated value of *a*. Thus, variables can be used to quickly control the shape of your curves.

Now, suppose you wanted to compare two plots? Gnuplot can do this, too. The range for the plots (the abscissae) will be identical, but additional functions can be given as long as those functions are separated by a comma. That is, plot will treat each expression between the commas as a new curve to display. For example, try typing the following:

```
plot [-1:1] x**2, x**4, x**6, x**8
```

These curves display the increasing "squareness" you would expect for a polynomial as you increase the exponent on the range [-1, 1].

*Other Plotting Features*

As we are primarily focused on the ability of Gnuplot to fit functions, we will only briefly touch on some of its other plotting features. You can find more details about the following topics on the web tutorials included at the beginning of this document.

- **Plot Title (Axis Labels, etc.)**

  You can give your plot a title using the `set title` command. This command appears on a separate line from the plot command, as demonstrated below.

  ```
  set title "Exploring Polynomials"
  plot [-5:5] x, x**2
  ```

  Many, many other options exist to modify the behavior of Gnuplot. And in general, if you were using Gnuplot to create publication-quality illustrations, you would use these options to customize your output. For example, you can use the following commands to display a plot of radioactive decay:

  ```
  set title "Radioactive Decay"
  set xlabel "Time (s)"
  set ylabel "Disintegrations"
  set yrange [0:60]
  set tics   out
  plot [0:500] 50*exp(-x/150)
  ```

- **Dataset labels**

  It's possible to change the labels for each dataset using the `title` keyword. For example, if you don't like the default behavior (which displays the function expression), you can type:

  ```
  plot [-5:5] x title "Linear", x**2 title "Quadratic"
  ```

  This is equivalent to typing:

  ```
  plot [-5:5] x t "Linear", x**2 t "Quadratic"
  ```

  Which is nice, because it gets old typing "title" over and over again. If you don't want any label, simply enter the empty string (`""`).

As we mentioned, there are many other options that can be manipulated within Gnuplot, but this should get you started toward making good, clear graphs.

*File Access and Saving Your Plots*

Now that we've discussed creating plots, we need to discuss how Gnuplot interacts with files on your computer. This is important for two reasons: (1) If you save your plot, you will need to know where it is saved, and (2) if you want to load some experimental data, Gnuplot needs to know where to look.

This problem is handled by adjusting Gnuplots "current directory." When you initially start Gnuplot, the program will look for files in the same folder where you placed your executable (normally `gnuplot/binary/`). But this isn't particularly convenient: most people like to keep their data in a different folder, e.g. a digital lab notebook, a poster directory, a directory for coursework, etc. Thus, we need to change the working directory for Gnuplot when we start working with files. While we can use the Gnuplot `cd` command to change the current working directory, in Windows it's much easier to do this graphically. If you click the "Change Directory" button ( ) at the top of your window, you will be asked to click on a new working directory. From then on, all files that you save will be written to the folder that you choose. Similarly, Gnuplot will look in that directory if you tell it to open a new file.

Now that we've discussed the current working directory, we can think about how to our plots. This is an important feature, because eventually you may want to use a plot in a document or a poster. In Gnuplot alternate output is managed using "terminals." The normal terminal corresponds to your screen, but you can use a postscript file or PNG file as a terminal too (as well as others). When the plot looks good on your screen, you simply change the terminal type, replot, and now you have a hard copy. As an example, here are the commands needed to create a hard copy of the radioactive decay curve above in PNG format:

```
set title "Radioactive Decay"
set xlabel "Time (s)"
set ylabel "Disintegrations"
set yrange [0:60]
set tics    out
plot [0:500] 50*exp(-x/150)

set terminal png
set output "plot.png"
replot
unset output
set terminal wxt
```

The five lines at the end tell Gnuplot to save a copy of the current graph to the file `plot.png` in the current working directory. If you wanted a PostScript file, you would simply replace `png` with `postscript`. The final two lines tell Gnuplot to close the output file and set the terminal back to the default value, wxt – handy commands to know, even if you don't fully understand what Gnuplot is doing behind the scenes.

It's helpful to know a little bit about the differences between PostScript and PNG files, but that's outside the realm of what we can cover here. Suffice it to say that for your general use, you should be able to paste PNG files into word documents if you need to.

*Using Gnuplot to Plot Points*

So far, we've looked at Gnuplot only as a tool for plotting continuous curves, but it's primary use for us here is to examine discrete points of data (and then to fit curves to the data). Many times, our data will be in text format. Consider the following text file, which contains the points of a line (`linear.txt`):

```
# Linear Data File
#  x      y    err
 1.0    2.3   0.3
 2.0    6.5   0.3
 3.0   10.8   0.3
 4.0   16.3   0.3
 5.0   20.8   0.3
 6.0   24.8   0.3
 7.0   29.4   0.3
 8.0   34.2   0.3
 9.0   39.4   0.3
10.0   42.9   0.3
11.0   47.5   0.3
12.0   51.9   0.3
13.0   56.7   0.3
```

The file contains three columns of data, $x$, $y$, and $\sigma_y$. These numbers could represent anything: kinetics data (where $x$ is time and $y$ is concentration), extinction coefficient data (where $x$ is concentration and $y$ is UV absorbance), thermodynamic data, etc. The data can be downloaded from the course website as `linear.txt`. In this case, the numbers are roughly linear, with a slope of 4.5 and an intercept of -2.0.

Gnuplot can read this file directly, which is convenient for us: It ignores lines beginning with hash marks (these are comments), and it splits the columns up based on the white space between numerical values. It can read both short files and long files, and needn't be bound by the conventions used here: For example, if your file uses % to comment lines, and numbers are separated by commas, you can adjust settings in Gnuplot to compensate. It's also trivial to re-format data files using utilities like `awk`.

Now the question becomes, how do we plot this data? If `linear.txt` is stored in your working directory, you can plot it using the following syntax:

```
plot [0:15] "linear.txt"
```

This command reads the data from the given file and displays it, assuming column 1 is $x$ and column 2 is $y$.

Currently we're using only two of the three columns, and to access the third column we must introduce the `using` keyword. This keyword is similar to the `title` keyword in that it modifies the plot statement, and it can also be abbreviated with `u` instead of `using`. We use `using` to designate the columns we want to plot. If we want to plot columns 1 and 2 as $x$, $y$, we can type:

```
plot [0:15] "linear.txt" using 1:2
```

This statement has identical behavior to the one we used previously – the default behavior is to use columns 1 and 2 as *x* and *y*, respectively. Alternatively, we could plot columns 1 and 3 together. This is less interesting, but it could help us to see if there were any systematic variation to our uncertainties if we hadn't set them explicitly:

```
plot [0:15] "linear.txt" using 1:3
```

If you have a large file containing many numbers, this syntax will be useful to select which columns you want to view.

But we still haven't answer the key question: how do we display error bars? Since we're spending so much time in class discussing the importance of uncertainties, it would be silly not to include error bars in this plot. The effect for this specific dataset isn't very helpful: the error bars are pretty small and can't be seen unless you zoom in to the plot. But it's still good to include error bars on most plots.

To display error bars, we use a "style" in Gnuplot. Styles control how points are displayed, and there are several styles that can be used. When a style is specified, more data columns may be needed: in this case we'll need to use all three columns of the data. The extra column can be specified with the `using` keyword, and the style itself is given with the `with` keyword, as shown here:

```
plot [0:15] "linear.txt" using 1:2:3 with yerrorbars
```

The `yerrorbars` style looks in the first `using` column for *x*, the second for *y*, and the third for $\sigma$. Thus, three columns must be specified when using this style.

When you plot this particular function, you will observe that the error bars are small, but they are indeed shown. Below is a plot I made with the following command, which can be used to artificially inflate the size of the error bars (by setting them equal to 5):

```
plot [0:15] "linear.txt" using 1:2:(5.0) with yerrorbars
```
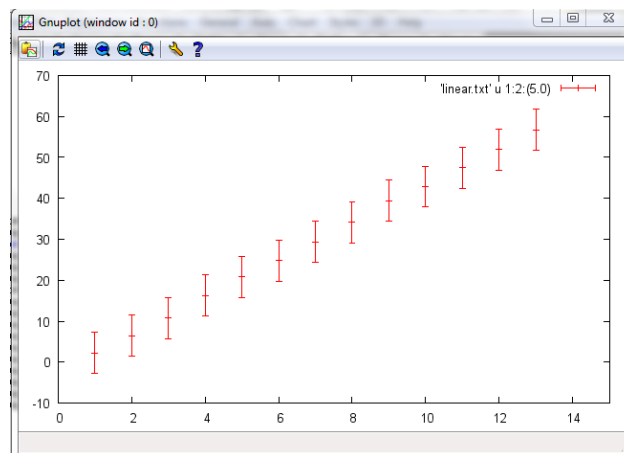
Other styles exist, and you can look them up online, but for our purposes, we will use the `yerrorbars` style most frequently.

*Fitting a Function*

Now that we've covered the basics, we can start discussing how to fit functions. Fortunately, it's pretty easy now that we've gotten this far.

Suppose I know the model I wish to fit to the data is a straight line. Given that we've looked at the data (and I've told you it's linear), that model makes sense. Moreover, there's an easy functional form: $y = mx + b$. We know the "true" values for $m$ and $b$ because this is artificial data. However, generally we wouldn't know these – we'd have to do the fit to estimate them.

We can proceed in Gnuplot using some simple commands. In fact, the following four lines take care of fitting for us:

```
f1(x) = m*x + b
m = 3.0
b = 1.0
fit f1(x) "linear.txt" using 1:2:3 via m, b
```

The first line defines our model: a simple linear equation in terms of $x$. The second two lines define initial estimates for our parameters. Looking at the data, we know that the slope is positive, and the intercept is somewhere near 0. In this particular case, we can be much more inaccurate in our guesses and still get reasonable results, but sometimes we might end up in a false minimum with the wrong guesses: the only way to know for sure is to plot the fit and see how it compares to our data.

The final line does the fitting. You must supply a function ( `f1(x)` ), data to use for the fit, and a definition of which columns correspond to $x$, $f(x)$, and $\sigma$. Finally, the `via` keyword tells Gnuplot which variables in the function are parameters, i.e. what to optimize.

When you type the commands above, you'll see some information as the program attempts to find the best parameters. When everything is finished, however, you should see the final summary of the fit (a copy is appended to the `fit.log` file in your working directory):

```
After 5 iterations the fit converged.
final sum of squares of residuals : 17.8071
rel. change during last iteration : -4.37708e-12

degrees of freedom    (FIT_NDF)                        : 11
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 1.27233
variance of residuals (reduced chisquare) = WSSR/ndf   : 1.61883
p-value of the Chisq distribution (FIT_P)              : 0.0861663

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
m               = 4.54176          +/- 0.02829      (0.623%)
b               = -2.29231         +/- 0.2246       (9.797%)
```

Some information is also given on the covariance matrix for the fit, which you can safely ignore for this tutorial. Obviously, there are times when you can't ignore this information, so keep that in mind if you are doing fits in the future.

The important information that you are interested in is the values of m and b (along with their estimated uncertainties), as well as the reduced chi-square ($\chi^2/v$). As you can see in this example, the parameters are within error of what I used to generate the plot, and the reduced chi-square is about equal to one. This is a nice fit.

One important thing to know is that the fit function will work without uncertainties. If you specify a file without uncertainties, it will assign a σ of 1 to all of your y data points so they are all equally weighted in the fit. This will probably give you a good fit, but you must remember that **the uncertainties on the fit parameters are meaningless in this situation. Parameter uncertainties are only meaningful when experimental uncertainties are provided!**

When the fitting has finished, Gnuplot conveniently updates the values of *m* and *b* for you. Thus, if you want to visualize your fit, you can use the following command:

```
plot f1(x), "linear.txt" with yerrorbars
```

If everything has worked properly, the curve will pass very close to the experimental data points.

*Plotting Residuals*

As discussed in class, one of the most important criteria for assessing the quality of a fit is a plot of the residuals. Even if the uncertainties on the parameters are small and the reduced chi-squared value is reasonable, the residuals may still indicate a poor fit. Remember, to pass muster the residuals must be small (comparable to the uncertainty in the data) *and* randomly-distributed.

Gnuplot allows you to examine the residuals using a simple syntax.

```
set xzeroaxis lt -1
plot "linear.txt" using 1:(f1($1)-$2):3 w yerrorbars
```

In these commands, the first line is simply a setting that tells Gnuplot to display the zero axis on the plot (essentially the line $y = 0$). The second is more interesting: we are plotting linear.txt, but we are using the `using` keyword to "trick" Gnuplot into manipulating our data before displaying it. The first and third columns are what we'd normally expect: use the *x* values from the file, and use the error bars, too. The second column tells Gnuplot to use the difference between $f(x_i)$ (`f1($1)`) and $y_i$ (`$2`) as the ordinate of the plot. The dollar signs reference the columns themselves from within a `using` statement, so e.g. `$2` refers to column two in the calculation. The parenthesis are needed for this syntax.

This method for plotting the residuals is more than adequate for this course; however, you may in the future want to try something a little more elegant. The following commands display the residuals and the fit function on the same plot, using the right-hand axis for the residuals. I'm not going to go into what each command does – hopefully you can follow most of it from the tutorial

so far. However, I will point out that in the final plot command, a backslash (\) is used to continue from one line to the next. There is nothing following the backslash, it should be the last character before the end of the line. If there's a space or another character, the line won't continue, and Gnuplot will register a syntax error.

```
reset
set title    "Linear Fit with Residuals"
set xlabel   "Time (s)"
set ylabel   "Observation"
set y2label  "Residuals" rotate by -90
set yrange   [-20:60]
set y2range  [-1:10]
set tics     out
set ytics    nomirror
set y2tics   border
set x2zeroaxis lt -1
plot "linear.txt" w yerrorbars t "", f1(x) t "", \
     "linear.txt" using 1:(f1($1)-$2):3 w yerrorbars axes x1y2 t ""
```

Below is the resulting graph. I wouldn't quite call this publication quality, but it's certainly getting there.
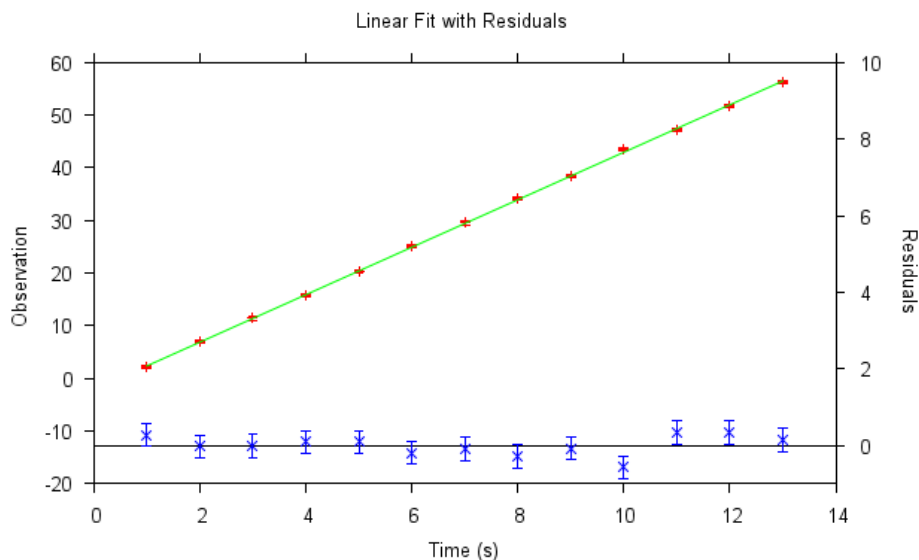


*Editing Longer Plots*

As you can see from the previous example, sometimes plot definitions can be rather long and involved. And from a science perspective, it's always good to keep a log of exactly what commands you used to create a particular graph. To that end, often it's better to write your graph commands in a separate program (like Windows Wordpad or a more advanced editor like Notepad++, Atom, vi, or Xemacs). Then, when you've saved your commands, you can load your plot into Gnuplot without having to retype everything.

Gnuplot provides several ways to read plot data. First, you can simply open the file from the menu bar at the top of the window. Alternatively, you can read a list of commands from the command line. If you have written some plot commands into a file called `test.plt` (in the current working directory), and you type:

```
load "test.plt"
```

Gnuplot will happily read the commands you've written in `test.plt` and execute them line-by-line.

Alternatively, you can save the history of commands you've written. This history will contain a lot of additional commands that represent the Gnuplot defaults, but it should represent a complete (if cluttered) history of your Gnuplot session. To save your plot, simply type

```
save "test.plt"
```

This command will save the session details into the given file (in this case `test.plt`). As always, you can open this file in a text editor and look at exactly what commands are stored there.

*Conclusion*

I've mentioned numerous times in this tutorial that Gnuplot is a powerful tool, and my goal with this document is to show you some examples of how to use the program to analyze data and model fitting problems. There are many more features in Gnuplot that we haven't discussed, but hopefully you are knowledgeable enough to investigate those on your own after reading this far.

If you wish to read more, the help documentation for Gnuplot is very useful, and the online manual is also very informative. Help is available within the program by typing `help <keyword>`, and the manual is available at:

http://www.gnuplot.info/documentation.html

Here's to successful fitting!