Plotting Data with gnuplot

This tutorial is intended as a supplement to the information contained on the Physics' Department website: Plotting and Fitting Data and Plotting Data with Kaleidagraph. It shows how to perform the same functions described in those tutorials using gnuplot, a command-line-driven plotting program commonly available on Unix machines (though available for other platforms as well). You may find it helpful to look at the other tutorials as well; this one is intended to follow them quite closely.

The instructions and samples given correspond to version 3.7 running under Linux, but the results should be similar everywhere. If you are using an older version, however, you might find a few of the more advanced features missing.

Introduction

gnuplot seems almost the antithesis of Kaleidagraph: the the Kaleidagraph tutorial calls Kaleidagraph "an easy-to-use if somewhat limited graphics program". gnuplot is a not-quite-as-easy-to use, though extremely powerful, command-line plotting program.

Running gnuplot is easy: from a command prompt on any system, type `gnuplot`. It is even possible to do this over a telnet or ssh connection, and preview the graphs in text mode! For best results, however, you should run gnuplot from within X Window, so that you can see better previews of your plots.

Entering Data

All the data sets you use in gnuplot should be typed into a text file first. There should be one data point per line. Each data point will consist of several numbers: the independent variable, the dependent variable, and optionally error bars. Each of these fields should be separated by a tab.

Actually, any number of fields may be specified on each line; this is useful if you have multiple measurements for each data point, for instance. For information about how to access this additional information in your plots, see (fixme: add section) below.

You may include any extra information you want in the file, such as a description of the data, headings for each of the data columns, and so on, as long as each such line begins with the comment character, `#`.

The dataset used in this example is available in the file cavendish.data.

Plotting Functions

Basic Plotting

Plotting functions in gnuplot is really quite easy. Suppose you want to plot the function f(x) = exp(-x^2 / 2). In gnuplot, exponentiation uses `**`, not `^`. So, after starting up gnuplot, at the `gnuplot>` prompt you would type:

```
plot exp(-x**2 / 2)
```

Usually, you'll want a little more control over your plot, at least specifying the ranges for the x- and y-axes. You can specify these in a **[minimum:maximum]** form before the function. Specify the x range first, then the y range. You may leave off the y range, or both. We can revise our previous plot command to:

```
plot [-4:4] exp(-x**2 / 2)
```

Here, the y range will be automatically determined.

More Function Plotting

If you'd like to plot more than one function, simply list all the functions, separated by commas. For example:

```
plot [-4:4] exp(-x**2 / 2), x**2 / 16
```

You can also rename the independent variable, if you'd like. This is done in conjuction with specifying the plot range:

```
plot [t=-4:4] exp(-t**2 / 2), t**2 / 16
```

Defining Functions

Sometimes, it may be convenient to define a function, so that it does not have to be retyped each time. It's easy to do this. Let's create a function `f(x)` to represent our bell curve, then use it in the plot:

```
f(x) = exp(-x**2 / 2)
plot [t=-4:4] f(t), t**2 / 16
```

Plot Titles, Labels, and Legends

You might have noticed that when you produced your plots, a legend was automatically created in the upper-right corner of the plot. By default, the name of each curve is simply the formula you typed in. You can give them other names using the **title** attribute. Simply follow the formula for the function with **title "The Title"**. We can change our previous command to:

```
plot [t=-4:4] f(t) title "Bell Curve", t**2 / 16 title "Parabola"
```

Note that commas are never used except to separate distinct functions. If you would like a curve not to show up in the legend, set its title to `""`.

We can also add a title to our plot, and some labels on the axes. The **set** command is used. Here is an example:

```
set title "Some Sample Plots"
set xlabel "Independent Variable (no units)"
set ylabel "Dependent Variable (no units)"
```

These changes do not have an effect until you redraw the plot. This can be done by typing in the plot command again, but if the plot itself does not change, it is enough to type:

```
replot
```

to replot the last functions given.

Other Nice Touches

It's often nice to add a grid to the plot, making it easier to see where functions and data fall on the plot. To do this, type

```
set grid
```

Plotting Data

After learning how to plot functions, now it's time to learn how to plot data. The syntax is almost the same, except that instead of specifying a function, you must specify the name of the file containing the data to plot, enclosed in double quotes.

First, since we were playing around with plots above, we'll clear the labels on the axes and other settings:

```
reset
```

You could also quit gnuplot and restart it for the same effect. Now, we'll plot our sample data. We'll get rid of an entry in the legend for our data points by using a title of `""`, and also set up the axes on the plot:

```
set title "Cavendish Data"
set xlabel "Time (s)"
set ylabel "Angle (mrad)"
set grid
plot "cavendish.data" title ""
```

When plotting data, it usually isn't necessary to specify ranges for the independent and dependent variables, since they can be figured out from the input. If necessary, of course, you can always give them.

Including Error Bars

Since our input data contains uncertainties for the measured (dependent) quantity, we can create y error bars. Once again, gnuplot makes this easy:

```
plot "cavendish.data" title "" with yerrorbars
```

It's possible to plot x error bars using `xerrorbars`, or both x and y errorbars using `xyerrorbars`. When both x and y error bars are used, there must be four columns present, and x error bars must be specified first in the data file. More variations are possible; see the online help for more information.

Note that gnuplot can be very picky about the order you give modifiers to the plots. If you were instead to type

```
plot "cavendish.data" with yerrorbars title ""
```

You'd get a rather strange error message: `undefined variable: title`. If you see error messages like this, check the ordering in your plot commands. If you're unsure, typing `help plot` should get you straightened out.

Fitting Data

No plotting program would be complete without the ability to fit our data to a curve. For the Cavendish experiment, we'll need to fit our data to a sinusoidal curve with exponential decay. gnuplot supports these nonlinear curve fits, and can even take the experimental uncertainties of the data points into account.

First, it's necessary to define the form of the function we'll try to fit to. Define this as you would any other function in gnuplot, but leave variables for the fitting parameters. We'll use

```
theta(t) = theta0 + a * exp(-t / tau) * sin(2 * pi * t / T + phi)
```

The various fitting parameters are:

- `a`: the amplitude of oscillations
- `tau`: the period of oscillation
- `phi`: the initial phase
- `T`: exponential decay time
- `theta0`: shift from equilibrium position

For a non-linear curve fit such as this, it is often necessary to provide an initial guess for each of the fitting parameters, or the fitting attempt

may fail. For simple equations, such as polynomials, this will not be necessary (but never hurts).

```
a = 40
tau = 15
phi = -0.5
T = 15
theta0 = 10
```

Finally, we'll do the actual curve fit. The syntax for this is:

```
fit theta(x) "cavendish.data" using 1:2:3 via a, tau, phi, T, theta0
```

Here's how the command is interpreted: `fit` tells gnuplot we're doing a curve fit. The next part, `theta(x)`, must be a function that we're using to fit the data. Here we must use x as the independent variable. The next part, `"cavendish.data"`, must be a datafile containing the data we wish to fit. The `using 1:2:3` tells gnuplot to take columns 1, 2, and 3 from the data file and use them as the x, y, and uncertainties, respectively. If this part is left out, then the experimental uncertainties will not be used for the curve fit. See below for a greater discussion of the extremely powerful **using** qualifier. Finally, we must tell gnuplot what variables it can adjust to get a better fit. For this case, we say `via a, tau, phi, T, theta0`.

gnuplot will produce output as it proceeds through the fit, and if the fit is successful you should see something like this:

```
degrees of freedom (ndf) : 34
rms of residuals      (stdfit) = sqrt(WSSR/ndf)      : 1.07102
variance of residuals (reduced chisquare) = WSSR/ndf : 1.14708

Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 44.5389          +/- 2.127        (4.776%)
tau             = 57.5667          +/- 8.132        (14.13%)
phi             = -0.377254        +/- 0.04235      (11.22%)
T               = 13.1026          +/- 0.06465      (0.4934%)
theta0          = 2.45704          +/- 0.6081       (24.75%)


correlation matrix of the fit parameters:

                a      tau    phi    T      theta0
a               1.000
tau            -0.844  1.000
phi            -0.100  0.088  1.000
T              -0.072  0.072  0.806  1.000
theta0         -0.166  0.127 -0.182 -0.166  1.000
```

Important quantities to note are the reduced chi square (variance of residuals), which in this case is 1.15, and the values for each of the fitting parameters. Each fitting parameter also has an uncertainty listed. The correlation matrix at the end can ususually be ignored.

gnuplot has also stored the fitting parameters in the variables, so producing a plot with the associated best-fit curve is as easy as:

```
plot "cavendish.data" title "" with yerrorbars, theta(x) title "Best-Fit Curve"
```

More Advanced Features

Using `using`

The `using` qualifier used in the fitting command above is an extremely powerful tool in gnuplot. With it, you can exercise almost limitless control over your data as it is plotted.

gnuplot usually expects a certain number of data columns when it uses a data file (whether for plotting or fitting). Usually, only two columns are used: the independent variable and the dependent variable. With error bars, one or two more columns may be used. Usually, these columns are taken out of the datafile directly. Sometimes, it's necessary to exercise a little more control. That's where `using` comes in.

Say you need to swap the two data columns, since that the dependent variable comes first, followed by the independent variable in the data file. You can produce this plot with the command:

```
plot "reversed.data" using 2:1
```

The `using` command expects several values, one for each column of data required, with each value separated by a colon. If the value is simply a number, gnuplot will take that data piece from the specified column in the datafile. In this case, we tell gnuplot to take the independent variable from column 2, and the dependent variable from column 1.

The previous example was a bit contrived. But there's a very common case where `using` is used: when there are multiple data sets in an input. Suppose you have a datafile with three columns: an independent variable, and two dependent variables. You'd like to plot both dependet variables as a separate set of points. You can use:

```
plot "double.data" using 1:2 title "Series 1", "double.data" using 1:3 title "Series 2"
```

In our fitting example above, by specifying `using 1:2:3`, we were forcing the fit command to take three columns as input, instead of the usual two (to include the error information), but we did not perform any reordering on them.

This is still just scratching the surface of what `using` can do. Instead of giving a column number, you can also specify a complete expression, which must be surrounded in parentheses. Within this expression, the values from the columns can be accessed as `$1`, `$2`, `$3`, etc. As an example, if we wanted to plot the natural logarithm of our dependent variable, we could use:

```
plot "log.data" using 1:(log($2))
```

Note that as part of a `using` qualifier, `($2)` is exactly equivalent to `2`.

For another example, see the next section: plotting residuals.

Plotting Residuals

To understand this section, you'll need to have understood the section "Using `using`" above.

First, we'll produce a plot of the difference between each data point and the fitted curve:

```
plot "cavendish.data" using 1:(theta($1) - $2):3 title "Residuals" with yerrorbars
```

A little explanation of the using statement is perhaps in order. We're producing a plot with y error bars, so we need three data columns. Hence, the `using` qualifier has three parts, separated by colons. The first, `1`, says the first part, the independent variable, is simply the first column from the input file. The second part is an expression: we'll compute the difference between our function (`theta`), evaluated at the value for the independent variable (`$1` - first column of data file), and the measured value (`$2` - second column of data file, or dependent variable) for that point. The third column, `3`, simply says to use the existing uncertainty stored in column 3 of the data file with no modification.

It would be even better if we could put the residuals on the same graph as the fitted curve. To make this look good, we'll use a different scale for the residuals, so they can be separated from the rest of the graph. gnuplot allows you to use two different scales for each axis: there are independent x and x2 scales for the x-axis, y and y2 scales for the y-axis, etc.

There's another syntax for defining the ranges for each of the axes, which is necessary for using more than one scale at a time. First, let's shift the graph of our data and fitted curve up a bit, to make room.

```
set yrange [-80:60]
plot "cavendish.data" title "" with yerrorbars, theta(x) title ""
```

This is like specifying the range as part of the plot command, but the settings will stick around until they are overridden, and we can specify a y-range without an x-range.

Now, we'll create a second scale for the y-axis on the right-hand side. This can be accomplished by:

```
set y2range [-20:120]
set y2tics border
```

The `set y2tics border` command tells gnuplot to display this scale on the border of the plot. Without it, the new scale would be set, but it would not be shown on the right-hand side of the plot.

Now it's time to add our residuals. We add them to the plot command, and specify that they should use the new y scale. They will use the same x scale as before:

```
plot "cavendish.data" title "" with yerrorbars, theta(x) title "", "cavendish.data" using 1:(theta($1) - $2):3 axes x1y2 title "" with yerrorba
```

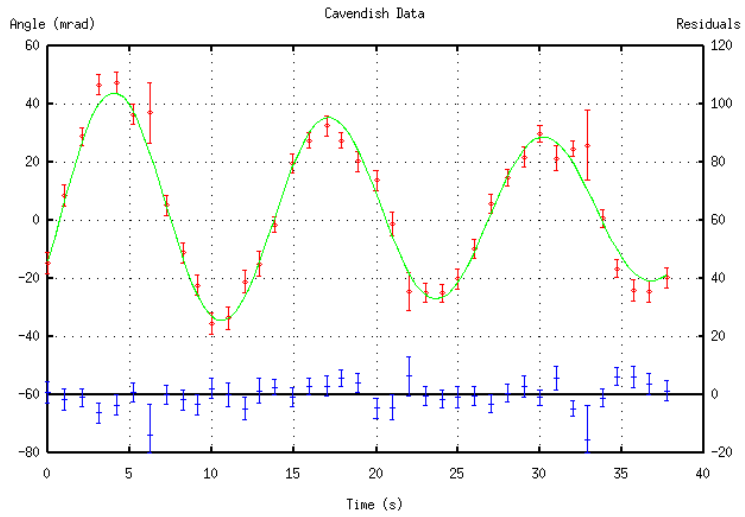
Here, `axes x1y2` means to use the normal x-axis, and the new y axis we just defined.

Finally, for a little extra touch, let's draw an x-axis for the residuals:

```
set x2zeroaxis lt -1
set y2label "Residuals"
replot
```

Here, `lt -1` stands for "line type -1", where -1 is the style usually used for plot borders.

If you've been following along the entire tutorial, you should now have a plot window that looks something like this:



Producing Printed Output

gnuplot is very device-neutral: when producing your plots, it could care less whether it is producing a preview on an X Window display, an ASCII-art version for a terminal, or any other output form. The plot commands will all work the same. After getting your plot the way you like it,

however, you'll want to save it or print it out. To do so, you'll want to change the output format, which can be accomplished with the `set term` command.

When running in X, the default terminal type is `x11`. If, after saving output elsewhere, you want to preview output in X again, you'll want to type

```
set term x11
```

If you are working over telnet or where graphics are not available, you can type

```
set term dumb
```

to choose the dumb terminal. While not pretty, you can often get an idea what your plot looks like with this. The Cavendish data looks like this on a dumb terminal: