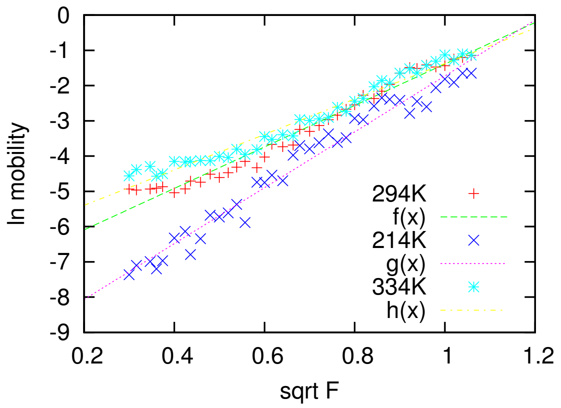
GNUPLOT - a powerful plotting program for professional scientific graphs

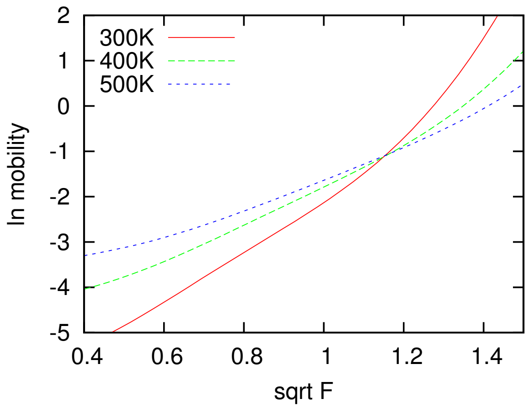
Sam Coveney
University of Sheffield Physics and Astronomy Department

Abstract

GNUPLOT is a command-line driven plotting program that is widely used in the scientific community. Professional graphs of publishable quality can be achieved with minimal effort from the user. The software is free and open-source, and can be easily downloaded and installed. GNUPLOT is also installed on the linux suite that can be accessed from university accommodations, the 3rd/4th year study room and the computer labs in Hicks.

The following graphs took a few minutes to produce, which included fitting functions to the data





Contents

1 Installing GNUPLOT

If you're using linux or a Mac, GNUPLOT can be downloaded and installed simply by typing the following into the command-line terminal:

```
sudo apt-get install gnuplot
```

If this doesn't 'find' gnuplot, you should try broadening the search of 'software sources', usually found under '/System/Administration', and then try again. If you use windows, you should visit this link and download the latest version of GNUPLOT for your operating system (typing gnuplot into google and following the appropriate links for download will quickly bring you to this page too):

http://sourceforge.net/projects/gnuplot/files/

Once downloaded, you will need to unpack the directory somewhere (right click on the zipped folder and click 'extract'). This will generate an unzipped folder, which you can put where you like. Open this folder, open the GNUPLOT sub-folder and then the 'binaries' folder, and you should find something called 'wgnuplot', an application. Make a short-cut and put it on your desktop, then you're ready to go.

GNUPLOT is much more difficult to properly install and use on windows, but we'll make do. I recommend partitioning some of your hard-drive for linux (Ubuntu for example) so that you can boot linux when you need to - this takes minutes to set up. For Mac users, I think 'MacPorts' lets you get GNUPLOT.

2 A few command-line tips

This section will provide a few command-line terminal operations which will allow you to use GNUPLOT. This information will be accurate for linux, and I'll include the commands for windows. The internet is your friend here. Fortunately there are very few things we need to know for our current purposes.

2.1 What is the terminal?

The terminal is just a way to do things on a computer that bypasses the graphical user interface, or GUI. A lot of good free software does not provide the user with a GUI, and instead requires the user to enter commands into the terminal; fortunately, this is very easy most of the time. The syntax is obvious, and if you make a mistake the terminal will usually point this out to you and suggest a correction.

You can usually find the terminal by looking in *Accessories*. Clicking on the icon will open up a black box. Alternatively, linux users can use *Ctrl-Alt-T* to open the terminal. You can fiddle with the colours and size of the terminal using the drop-down lists - white text on black background might look cool, but your eyes won't like it.

The terminal will prompt you for some sort of input, which should be typed and followed by pressing *enter*. When I open my terminal, I'm prompted with this

```
sam@sam-laptop: ~ $
```

2.2 Navigating your computer

It's best to think of the terminal as hovering over folders (known as directories in the terminal lingo). When you open the terminal, it will be hovering over your HOME directory. This will have a name like

```
/home/sam
```

So, what do I do to change the directory that I'm hovering over? I need to tell the terminal to **c**hange **d**irectory.

```
cd Documents/gnuplotlecture/
```

Careful, the command-line is case sensitive! We can usually get the terminal to complete what we are typing by pressing the *Tab* key. So if I typed **cd Doc** followed by *Tab*, the terminal would fill in the rest of the word; the word must not be ambiguous though, so **cd Do** would not be enough, since it could refer to either *Documents* or *Downloads*.

Also, using the up and down keys you can browse through previous commands. This is incredibly useful.

It's helpful to see what directories and files are contained in the current directory. To do this we simply type **ls** (or **dir** on windows).

```
sam@sam-laptop: ~ $ ls
```

If you want to go back a directory, enter **cd ..**

To get to the home directory, we can use the short-cut **cd** ~

This will probably be everything we need to use GNUPLOT.

2.3 Opening GNUPLOT

If I have a folder full of data files which I'd like plot with GNUPLOT, I would enter the following

```
cd Documents/folderwithdata/
gnuplot
```

This would open GNUPLOT in the terminal, and allow me to start plotting these data files. GNUPLOT looks for files, which I tell it to plot, in the current directory, which is why I first changed the directory.

Typing **help** into GNUPLOT will bring up large help menu. To exit GNUPLOT properly, type **exit**. When viewing the plotting window, press **q** to close it. Let's start learning about GNUPLOT!

2.4 windows

I've just got GNUPLOT to work on windows; it didn't take too long. That 'wgnuplot' short-cut I mentioned? Click on it and it'll open some sort of terminal-like window with GNUPLOT running in it. You'll see a tab at the top to change directory, so change it to the one with the data in. Then you can start plotting data from that directory, and any graphs you save will be saved in that directory too.

Even if you're not plotting data (and therefore assume you don't need to switch directory) please do! Otherwise your saved graphs will appear in the 'binaries' folder! That's not where you want them! Also, good luck getting the 'help' menu to appear. When I tried this, windows assumed I was asking for windows help, rather than for help from GNUPLOT.

3 Plotting with GNUPLOT

Plotting data is the most likely thing you'll want to use GNUPLOT for; we can produce graphs in a variety of formats, such as **.png**, **.eps** etc. Due to licensing reasons, **.pdf** files require an extra step - we should first make a **.eps** file in GNUPLOT, then convert it into a **.pdf** file externally (outside the program), which couldn't be easier; this process is described later.

The file types I've mentioned are postscript files. Before we learn how to produce them, we should first learn how to plot data. The standard output of GNUPLOT is sent to and displayed in the **wxt** terminal (**x11** terminal for Macs), which appears when you tell GNUPLOT to plot something, but disappears when you close the program.

3.1 Plotting basics

We've already opened up GNUPLOT, and the program is waiting for our input

```
gnuplot >
```

Before we learn how to plot data from files, lets just do some simple plotting. Type

```
plot sin(x)
```

And there you go. We may want to set the range of the the x or y axis

```
set xrange[0:5]
```

You'll notice that nothing has changed. You need to tell GNUPLOT to make another plot, which can be done by either re-entering the command to plot above, or by typing **replot**.

We may also wish to move the key somewhere else

```
set key bottom left
```

How about some labels for our axis?

```
set xlabel “x”
set ylabel “y = sin(x)”
```

A title for our plot perhaps?

```
set title “This is a plot of sin(x)”
```

We can change the tics on our axes easily too

```
set xtics 0.5
```

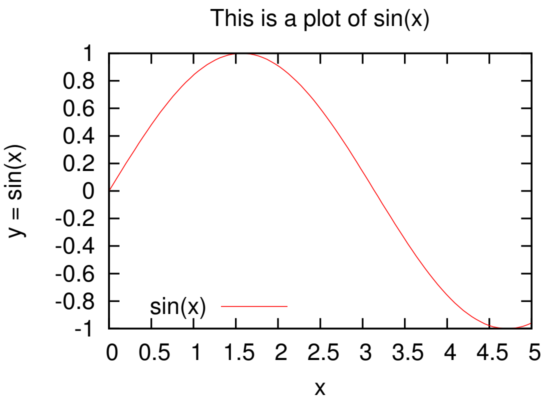
And here's the result, effortless and beautiful.

3.2 Plotting data from files

Firstly, what format is the data stored in? Are columns of data separated by whitespace (GNUPLOT assumes this by default) or by commas? There are of course other ways to separate data, such as with colons, but whitespace and commas are standard.

Let's assume that we have a file, **data.dat**, which has 5 columns of data, with each column on any row separated by a comma. It's traditional to use the file extension **.csv**, but it really doesn't matter what you use. So, we should tell GNUPLOT what separates data

```
set datafile separator “,”
```



By the way, we could have typed the following abbreviation for the same result

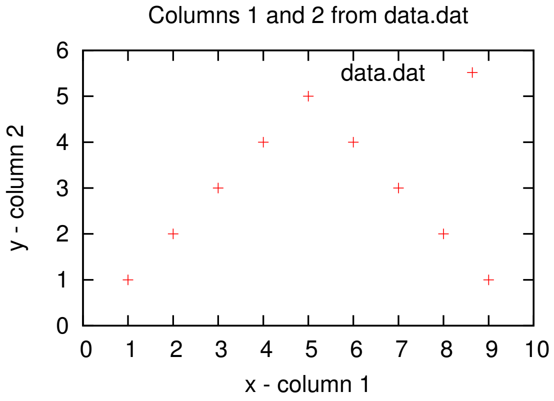
```
set dataf sep “,”
```

GNUPLOT allows you to abbreviate anything, so long as the abbreviation is not ambiguous.

So let's plot the first two columns of data against each other, columns 1 and 2 on the x and y axis respectively

```
plot “data.dat” using ($1) : ($2) title “data.dat”
```

The brackets and $ are fail-safes to ensure GNUPLOT does what you want. I've included a title for the key/legend to prevent GNUPLOT giving the title *“data.dat” using* ($1) : ($2) which would look awful. Along with naming the axes and giving a title to the plot, here's the result



We can plot data in numerous ways, including using built in functions to do calculations, for example

```
plot “data.dat” using (log($1)) : (sqrt($2)) title “data1”, “data.dat” using (($5)∗
∗2) : (sin($4)) title “data2”, “otherdata.dat” using ($1∗$3)) : ($4/$5)) title “data3”
```

This produces one plot with three differently coloured sets of data. Notice that I only used the **plot** command once, and separated subsequent data sets with a comma.

3.3 Saving and Loading

Once you've sorted out your plot and have it just right, you'll want to save your progress

save "myplot"

This is especially useful if you want a template for your plots. The save command does more than record your plot; it essentially saves everything that you have done in your session that is required to reproduce the plot. It might be that I have another data file, **data2.dat**, which contains data that I'd like to plot in the same format.

Then I'd open up GNUPLOT in the appropriate directory (the one containing **data2.dat** *and* **myplot** *and* **data.dat**) and type

load "myplot"

This would load the plot created using **data.dat**. Remember, GNUPLOT looks for files in the current directory, so we must make sure they're there, else we'll get an error message. We could then enter something like

plot "data2.dat" using ($1) : ($3) **title "newdata"**

This will plot the appropriate data from **data2.dat** with all the set-up that was saved in **myplot**. You might need to change a few things, but this is generally quite a time-saver, especially if you've used a myriad of settings which we've barely touched upon.

4 Postscript Format

Okay, so we can produce plots, but they disappear! What if we want to use them in a latex document for example? It's incredibly easy to change the output; I've never found need to produce anything other than an **.eps** file (encapsulated postscript), which I can use immediately in a Latex document, or a **.pdf** file, which I make from converting my **.eps** file, for use in a pdflatex document.

To get a list of possible types of outputs, type

set terminal

Generally, the following will do you just fine

set terminal postscript eps enhanced color *or* **set term pos eps enh col**

We still have one more thing to do, and that's tell GNUPLOT what the output will be. Really, this is just telling it what name you want to give to the generated file, which you can call anything you like (here I've given it the **.eps** extension just to be complete)

set output "data.eps" *or* **set out "data.eps"**

Almost time! There is one problem with changing the type of output we want; the resulting plot has different proportions than the one that was being displayed before. Usually, the text is far too small. I find the following command usually gives great plots. It just resizes the x and y dimensions of the plot so that they're appropriate for the postscript file we produce

set size 0.5,0.5

Okay, now we give the plot command and we're done! Usually, we'll have sorted out our plot in the **wxt** terminal (the one that displays the plot as default), so now that we've changed the output type we'll just have to type **replot**.

As soon as you've made your plot, I'd recommend that you reset the terminal type and output to default, which saves you accidentally destroying your precious plot by using the plot command again.

```
set term wxt or for Macs set term x11
set out
```

If you need to make changes to the graph you just produced, I'd also suggest deleting the current copy - not doing so sometimes gives strange results. Also, you may find that you need to tell GNUPLOT what the output file should be called again, even if you didn't set it back to default.

What if you want a **.pdf** file? Well, first open another terminal (because we can't do this within GNUPLOT). If you're on linux, you can type

```
sudo apt-get install epstopdf
```

This will install software to turn an eps file into a pdf file. Then simply type

```
epstopdf “data.eps”
```

If you're using windows, I'd suggest searching the web for some free software to convert eps files to pdf files. It may be installed on your system already, but who knows.

Now you're done! This pdf plot is ready to be put straight into a latex document or the likes.

5 Advanced Features

Everything above probably covers what you'll need GNUPLOT for. But we'd like to be able to exploit more of its power. We can do a lot more than produce fancy-looking graphs. We'd like to be able to fit our data to some function for example, or have different data plotted on different axes, or maybe just add a smooth line to guide the eye through a set of data points.

5.1 Joining up data points

Actually, this is hardly an advanced feature, it's just too easy. The first way we can join up our data is with lines that connect every single point. This works well sometimes, but not for noisy data. Here's how

```
plot “T234.csv” using (sqrt($1)) : (log($5)) with lines title ”234K”
```

That's correct, we ask it to plot the data in the file T234.csv with lines, and that's what it does - lines instead of points. If you plot several different data sets on the same plot, all with lines, GNUPLOT will automatically make them all different for you. If you change the output to postscript and you're worried that the lines will look the same in black and white, then fear not! GNUPLOT assumes this, so will draw the lines in your postscript file with solid lines, dashes, dots etc so that they all look different.

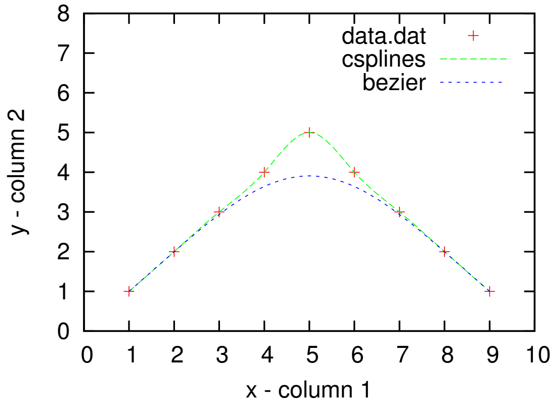
We can also get GNUPLOT to add smooth lines through our data, which is great if the data if noisy/messy and you want to show the general trend. To get a list of possible ways to add a smooth line to your data, provide GNUPLOT with some data like this

plot "T234.csv" using $(sqrt(\$1)) : (log(\$5))$ **title "234K" smooth**

The most useful of these two smoothing options are 'csplines', which joins up the points in your data smoothly, and 'bezier', which makes a smooth curve that doesn't necessarily trace through your data points. If I choose a smoothing option, but I want the points to show up on the plot too, then I have to plot the points explicitly by providing a plotting argument without asking for smoothing.

The following entry provides an example of these two smoothing options

plot "data.dat" using (\$1) : (\$2) **title "points", "data.dat" using** (\$1) : (\$2) **title "bezier" smooth bezier, "data.dat" using** (\$1) : (\$2) **title "csplines" smooth csplines**



The results are quite different, but I'm sure you'll be able to figure out which one to use when the time comes. Using the 'smooth' option in this way is a good way to help guide the eye through data. It probably shouldn't be used to demonstrate, say, a linear fit. Luckily, GNUPLOT provides a very easy way to fit functions to data.

5.2 Fitting Functions to data

Being able to fit data to functions can be vital if we are to make anything of our data. Luckily, GNUPLOT does this so easily you may think it read your mind and did it for you.

Firstly, we must be able to define a function. This is simple:

f(x) = a + b*x

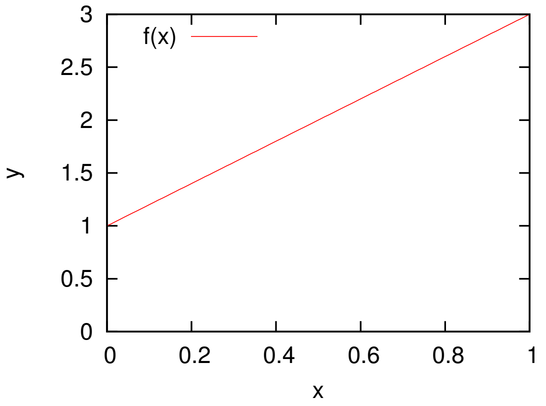
This speaks for itself. We could set 'a' and 'b' to some values as follows

a,b = 1,2 ***or*** **a = 1; b = 2**

We could also set 'a' and 'b' on two separate commands, but I've shown it this way to point out additional valid syntax.

plot f(x)

No surprises there. The function was defined exactly as we thought it would be. GNUPLOT allows you to define any function that you're capable of typing, if you get my meaning.

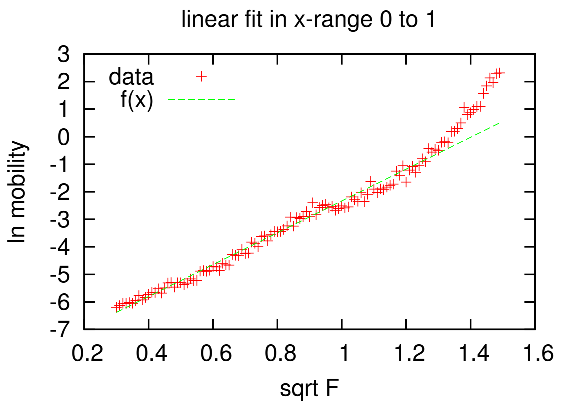


If you can type it, then most likely it'll be fine. Standard syntax like '**' for raising something to an exponent works fine, and use 'log()' when you require a natural log.

So what about fitting data? Well, I have some data I'd like to fit a linear function to, so I'll use $f(x)$. I can choose which parameters will be used for fitting using the **via** command. I can also define a specific range of x-values that I'd like to consider in the fit by stating these after the **fit** command (stating nothing causes all data to be considered for the fit).

fit [**0:1.0**] **f(x) "T274.csv" using** $(sqrt(\$1)):(log(\$5))$ **via a,b**

plot "T274.csv" using $(sqrt(\$1)):(log(\$5))$ **title "data", f(x) title "f(x)"**



I could have chosen to set 'a' to equal a constant first, and then used this last command, but fitting **via b** only. It is particularly important to hold some fitting parameters constant if you want to fit a function with many parameters and you already know what some of them are. It helps GNUPLOT to get the others correct. When you do fitting, GNUPLOT will inform you of the parameters it has calculated for your function.

Also, I made the fit using only two columns, sqrt(1) and log(5), which are x and y. I could have provided a third column, say column 6, with the error in y- values. Then my fit command would look something like this

fit [**0:1.0**] **f(x) "T274.csv" using** $(sqrt(\$1)):(log(\$5)):(\$6)$ **via a,b**

5.3 Axes Options

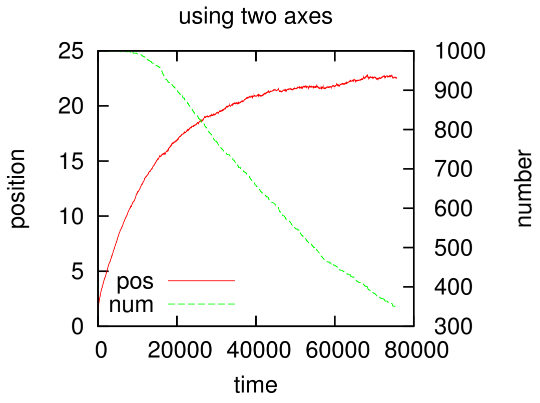
Sometimes you'll want to display two sets of data on a plot, but the scales of the data make this impossible to do on one axis. You should then just use two axes instead. You'll most likely want the same x-axis, but to plot something different on the left and right y-axes. Here's how

plot "moredata.csv" using ($1) : ($2) **with lines axis x1y1 title "pos", "moredata.csv" using** ($1) : ($4) **with lines axis x1y2 title "num"**

set ytics nomirror

set y2tics 100

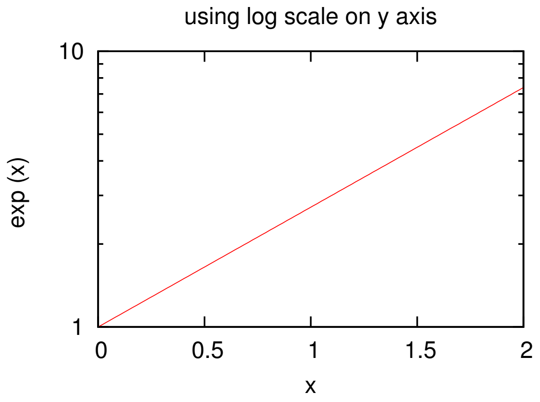
set y2label "number"



The **no mirror** command stops the tics on the left y-axis (y1) from being mirrored by the right y-axis (y2). But I must also tell GNUPLOT the tics that I want on the right y-axis, or it won't display anything there. Finally, I'd like to label the right y-axis. Done.

Using a log scale is another common feature you may wish to use. We just tell GNUPLOT we want to use a log scale on a particular axis, then make the plot

set log y



The other important thing that comes to mind is setting the zero axis, which means a dashed line will cross through zero on your chosen axis. This can be done for any axis.

```
set xzeroaxis
set y2zeroaxis
```

We may also want to set a grid. We can control this precisely, but it's usually enough for the grid to be defined by the tics on the axes.

```
set grid
```

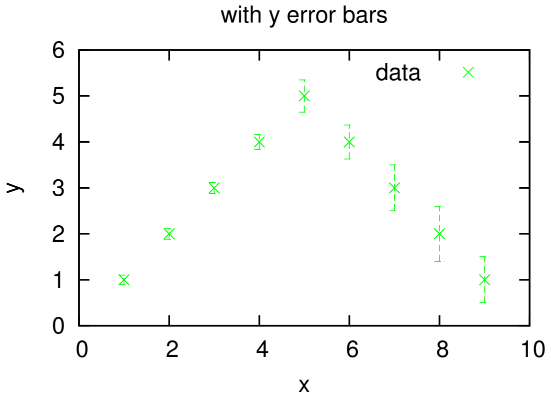
5.4 Error Bars

Error bars are pretty simple too. Remember how we could provide a third column of y-errors when fitting functions to data? Well we can tell GNUPLOT to include error bars in plots in nearly the same way

```
plot "data.dat" using 1:2:3 notitle with yerrorbars 2, "data.dat" using 1:2
title "data" with points 2
```

In the command above, I've told GNUPLOT to plot the data twice. Why? It just makes sure that I get the point symbol in my key to be *just the point*, rather than the point with an error bar attached. I've used some additional options, such as **points**, and there are two **2**'s floating around. This is just a style choice, which is covered next.

Before that, though, why didn't I include $'s next to my columns? It turns out that this isn't always necessary. I generally include them to be safe though.



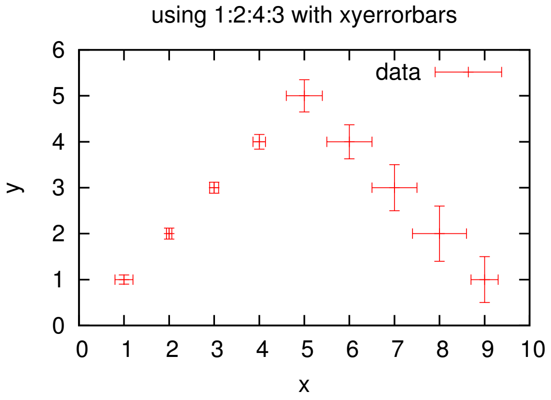
The next plot has both x and y error bars; the y-error was in columns 3 again, and the x-error in column 4. Note the key

```
plot "data.dat" using 1:2:4:3 with xyerrorbars title "data"
```

5.5 Styles

GNUPLOT generally does everything for you, but sometimes you'll want a bit of extra control over colours and line thickness. I'm not going to make a list of possible style options, but I'll just give a few examples of how to control the styles of lines and points.

You'll notice that the last command included syntax like **with 'something' 'number'**. Well this is the general command for controlling the style of your points, lines and error

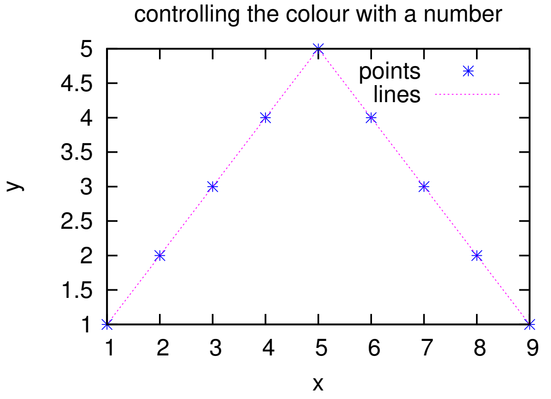


bars. There are a variety of other things you may want to check out; you can look at the options by entering, for example

plot "data.dat" using 1:2 with

This will bring up possible options. You can also type *help* into GNUPLOT to bring up a menu of help topics that include examples. Here's a simple example of choosing some colours

plot "data.dat" using 1:2 title "data" with points 3, "data.dat" using 1:2 with lines 4



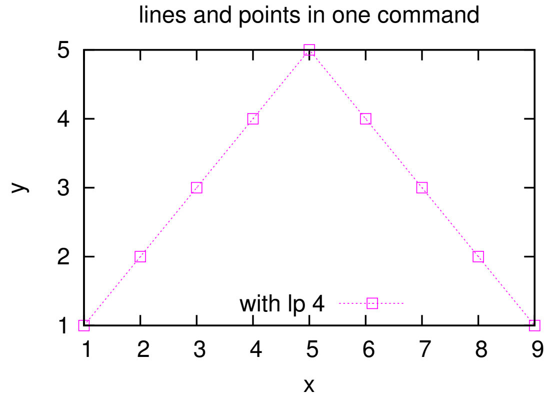
We can also use **with linespoints** to plot lines and points with one command, though this gives less control over style since they need to be the same colour. The syntax is

plot "data.dat" using 1:2 title "with lp 4" with lp 4

You may have noticed that I have been moving the key about. I can choose where to place the key using a command like **set key top left** or **set key bottom center**. I can also remove the key by using the command **unset key**.

5.6 Maths in your labels and titles

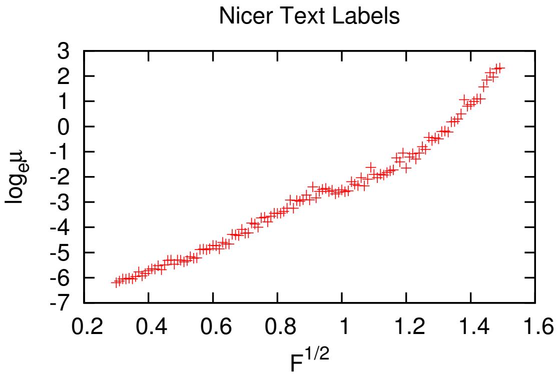
GNUPLOT does provide you with Greek letters and the ability to use exponents and under-



scores. However, this is *only possible in postscript format.* You can still label your graphs as I describe below whilst outputting to **wxt**, but the desired changes will only take effect when you output in postscript format.

It's best to look up a list of the Greek letters in the help menu or online. The general syntax for sub/super-scripts and Greek letters is

```
set xlabel "F^{1/2}"
set ylabel "log_{e}{/Symbol m}"
```



To use subscripts and superscripts at the same time, use this sort of syntax

```
"Z@^{2}_{64}"
```

6 Summary

This document is very brief, but if there's something you need to use which isn't included here then it's probably because I don't know how to do it. By using the **help** command in GNUPLOT, or by referring to tutorials and forums online, you should be able to work out how to do anything in GNUPLOT. There's a very short but possibly steep learning curve, but the effort this will take probably amounts to that required to rename your axes in Excel.

GNUPLOT is incredibly powerful. It can produce excellent 3D plots, plot surfaces in multiple colours, do any other types of plots you care to think of, and can even produce funky fractal patterns!

Hopefully I've convinced you that GNUPLOT is an excellent plotting program to use for any reports you are writing. It's very simple and effective. There are other plotting programs you could use, which might not be free, but which students can probably get at a discount. Perhaps ask around the department, CiCs or speak to the Free Software Society.

Don't leave it until write-up time to learn how to use plotting software.