Assignment 2: GnuPlot

Mikkel Bach Mortensen[1] Sarah Maria Niebe[2]
Gert Kruchov Sønderby[3]

[1] splab@diku.dk
[2] niebe@diku.dk
[3] atlas@diku.dk
Department of Computer Science, University of Copenhagen
Universitetsparken 1, DK-2100 Copenhagen East, Denmark

Abstract. This paper is intended as a beginner's guide to GnuPlot. As such it will contain instructions in the use of the software, and the first pitfalls that can be expected. It is not intended as an exhaustive reference.

1. Introduction

Plotting graphs and fitting functions to data points has many uses in computer science. Performance numbers such as CPU time or memory usage can be plotted to show the influences of various factors on the program. To create such plots, the plotting software GnuPlot is very useful. This paper is a short introduction to its use. It is highly recommended that this guide be supplemented with the GnuPlot reference [1].

2. Basic use

GnuPlot typically operates on data files consisting of tabulator- or space-separated values. Each line in the data file is considered to be a separate point along one axis of the plot, with entries on each line giving various values. The most basic data file for GnuPlot consists of one number per line.

To start GnuPlot, type `gnuplot` at the command line and press enter. The interactive interface should now start up.

2.1 Plotting

To plot data or a function in 2D, the `plot` function is used. At its most basic, the function is used as follows:

```
plot 'filename.data'
```

This produces a plot of the data file `filename.data`'s first column against the line number. However, if we have a two-column data file, the first column

being time, the second some data point, for example, we add the `using` predicate:

```
plot 'filename.data' using 1:2
```

Now, we have asked GnuPlot to use column 1 as the x axis, and column 2 as the y axis. It is possible to put up to four arguments in a `using` predicate, following this pattern:

```
x-axis[:x-error]:y-axis[:y-error]
```

If three arguments are given, it is assumed to be the two axes and the y-axis error value.

Columns are one-indexed, but column 0 is the line number itself.

The `every` predicate can be used to limit the amount of data used from the file. It takes up to six arguments, the full explanation for which can be found in [1]. A few illustrative examples are offered here, though:

`plot 'filename.data' using 1:2 every 3` will plot the data from the given file, but only every third line of data.

`plot 'filename.data' using 1:2 every 4::2` will plot the data from the given file, at every fourth line of data, starting with line number 2.

`plot 'filename.data' using 1:2 every 5::4::184` will plot the data from the file, using every fifth line, starting at line 4 and ending with line 184.

Line numbers are zero-indexed.

Instead of a data file, a function can be plotted, as well. In this case, simply insert the function name like so:

```
plot f(x)
```

To plot multiple items, separate them with commas, as in the following example.

```
plot f(x), 'filename.data' using 1:2 every 3
```

Further keywords of use include `with`, for changing styles of output. The reader is referred to [1] for a thorough reference to this and other keywords.

The command `splot` can be used to create three-dimensional plots. However, this is beyond the scope of this guide, and the reader is referred to [1] for further information.

2.2 Fitting

To fit a function to a dataset, the `fit` command is used. Start by defining a function, such as this example:

```
f(x) = a*x**2 + b*x + c
```

This is the function $f(x) = ax^2 + bx + c$. `**` is the power operator.

Now, if we have a dataset in *filename.data* that seems to follow this curve, and wish to fit the two, we can start by setting the variables a, b and c to reasonable values, and plot the data and function together. This should allow us to fit the function somewhat closely, but it is a lot of work, and not very exact.

If, however, we give the following command, with a, b, and c set to reasonable starting values, the result is far better:

```
fit f(x) 'filename.data' using 1:2 via a, b, c
```

Here we ask the program to fit the function f(x) to the data file by modifying a, b and c until it achieves the best fit. The data file can be given `using` or `every` predicates just as with `plot`. While performing this task, GnuPlot will output its values for the variables and how large an error margin the fit has for each iteration – this is also logged in the file `fit.log`. Once it is done, the variables will be set to the found values, so simply plotting the function will give the fitted version.

2.3 Outputting

By default, GnuPlot will output its graphs to the screen. However, it can be set to different media, allowing output to files, or directly to a printer.

To change the terminal mode (i.e. the output form) of GnuPlot, use the `set terminal` command. This command can take either `push` or `pop` as argument, or one of a long list of terminal types. The `push` argument will push the current terminal setup onto a stack for storage, while `pop` will reinstate the topmost setup on that stack. This can be used to avoid losing a certain setup when switching to another.

Of the many terminal types available, only a few are of interest to most users. The `postscript` type will output graphs as PostScript format, for use with various text systems, e.g. LaTeX, or for direct output to a PostScript printer. The `png` type will output a Portable Network Graphics image, and is suitable for files for use on the World Wide Web. The `fig` terminal type will give output in the Fig graphics language, which can be used with, among others, XFig. For further information on settings and special requirements for these, please see [1].

However, changing the terminal type is not enough, we also need to redirect the output. Redirecting this to a file is done by the following command:

```
set output 'filename'
```

If no filename is given, the default output (the screen) is selected. Otherwise, plots will be sent to the given file. Direct redirection to a printer can be done on Unix or related systems by giving the file name as `'|lpr'`. The normal options for lpr apply – it is run as a normal command line pipe. On MS-DOS and derived systems (i.e. Microsoft Windows®), giving `'PRN'` as the output target will send it to the default printer.

3. Pitfalls and Advice

GnuPlot is not a mathematical toolkit – it will not modify or simplify functions it is given. So in the interest of running time and fitting ability, any functions given should be as simple as possible. Using Maple or Mathematica to achieve this can be helpful.

Keep track of your terminal type and output location. Use the push and pop keywords for the terminal type, and reset output when you are done

outputting the plot you need. This way, you avoid overwriting plots, or wasting prints.

Keep an eye on the operators in your functions. A double-star exponent operator where a single-star multiplication is wanted, or vice versa, will completely mess up your function curve. So if your fit looks weird, start looking there first.

4. Conclusion

Accurate and easily understood plots can clarify much in computer science and other disciplines. GnuPlot is a powerful and simple tool, which will produce clear and concise graphs easily and with great detail. As such, it is highly recommended to users who need plots and graphs in their work.

References

[1] D. Crawford, Gnuplot: An interactive plotting program, Worldwide Web Document (1998). Available at `http://www.gnuplot.info/docs_4.0/gnuplot.html`.