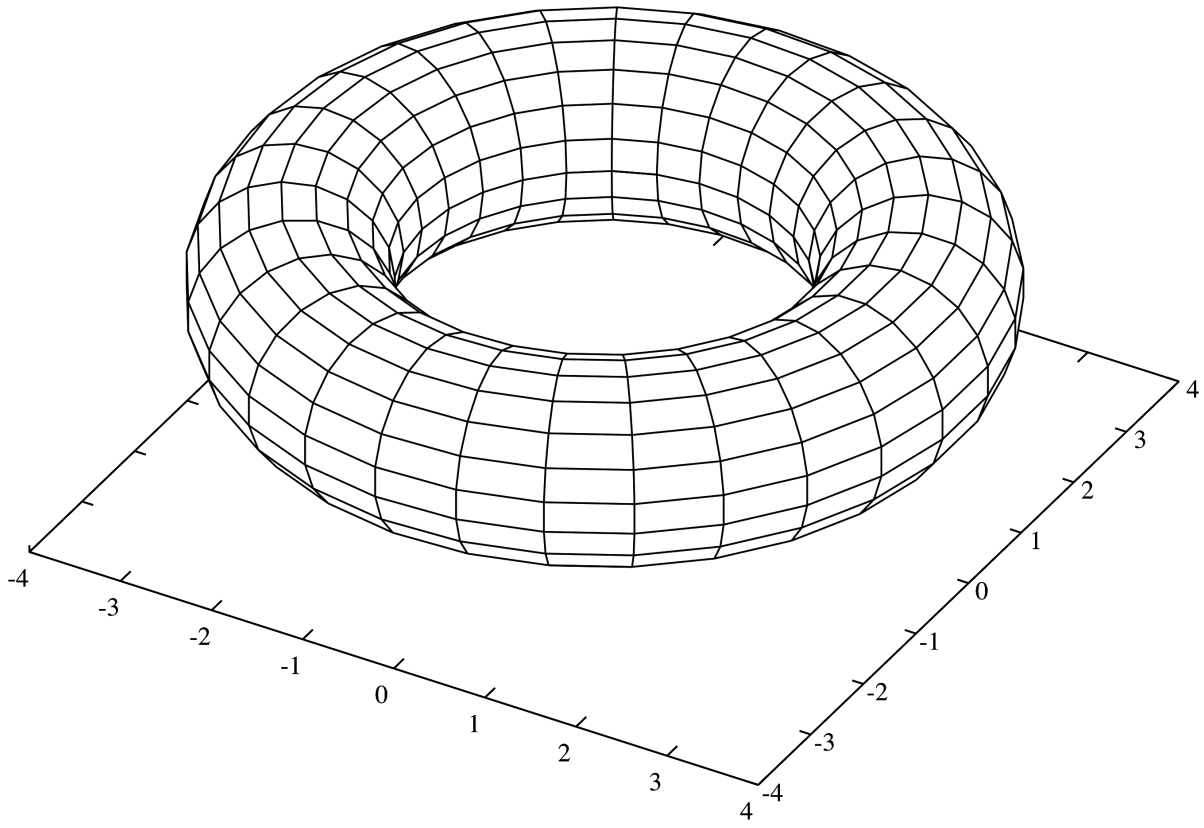
the pocket guide to

gnuplot



V.H. Belvadi

A note from the author

This guide has curious but sincere origins. It was written following the publication of an article[1] on installing gnuplot on macOS (then OS X), outlining a process which was not as straightforward as it should have been. The article became unusually popular, at least in terms of how many other sites, especially academic ones, linked back and sent traffic to it, and it soon became apparent that most people interested in learning gnuplot could probably do with a handy guide dedicated to helping them get started with the program.

Two other motivations exist for this: the first is my own learning of gnuplot for no reason other than the fact that it excited me, and the second is that it was part of the postgraduate course at my institution and I thought having an in-house manual would probably make things convenient for students.

Time is a teacher. By the time this short guide had been completed the meaninglessness of an in-house manual in a connected world had become clear and the book was simply made available publicly for anyone interested in making use of it. It was designed to get you started with gnuplot and not to master it. In fact there is no such thing as absolute mastery: if you can do what you set out to do with gnuplot then you have mastered it in that scenario. But, sooner or later, you will come across a highly specific requirement that makes you stop and question your entire understanding of a subject.

This guide is extremely short for a reason: it was meant to be in your pocket as a constant help while you started learning gnuplot and, like a couple of other introductory courses I have designed for the postgraduate level (e.g. C for physicists, LaTeX), the purpose of this book is to teach you enough gnuplot to give you a good grasp of it, and enough knowledge, by any count, to start teaching yourself the rest (and other similar programs). After all, where is the fun in not being self-taught in at least some disciplines?

the pocke guide to **gnuplot**

First edition, 2015.
Second edition, 2017.

Typeset in *Baskervald* and Nimbus Sans.

E-mail the author at vh@belvadi.com with your
comments and suggestions or for clarifications.

[1] https://vhbelvadi.com/installing-gnuplot-on-mac

What is gnuplot?

Plotting and analysing experimental data is a major part of physics and gnuplot is one of the most important, and among the oldest, tools used for this. It is a command-line driven graphing application available on nearly all platforms and can be used to draw extremely high quality precise graphs from both functions and data points with extremely simple commands.

Download gnuplot and set it up on your system using the appropriate executable from `www.gnuplot.info`. Installing on macOS is simpler if done via the Terminal. For macOS follow this article: `vhbelvadi.com/installing-gnuplot-on-mac/`.

Needless to say you have to work on gnuplot on the Terminal. It can, however, communicate with appropriate data and text files carrying space-separated tables. It outputs as well (and not in the Terminal) to several formats: png, eps, pdf, jpeg, svg etc. Therefore, even if an example in this guide uses a particular format you can replace it with any other without trouble.

Note that throughout this guide code is printed in a `monospaced font` and arguments (where custom data must be input based on your requirements) are mentioned within ⟨`angle brackets`⟩ so make sure that, while working in the Terminal, you do not type the brackets themselves, just the text they contain.

Once it has been installed on your system, begin accessing gnuplot by simply typing `gnuplot` on screen. It should take you to the gnuplot environment and start up its default terminal (i.e. an interactive display environment) for you. To set your own terminal use the `set terminal` ⟨`name`⟩ command. where ⟨`name`⟩ is the terminal you wish to use. You can use `x11` or `aquaTerm`. Try this command as soon as you enter gnuplot:

```
set terminal postscript eps enhanced
```

Discussing the specific capabilities of each terminal beyond this point is outside the scope of this guide. Once you are done working with gnuplot simply type `exit` to quit gnuplot and return to your standard Terminal prompt, usually the dollar prompt. By default gnuplot does not save outputs; this is partly to ensure test renders are not saved over and over again. So make sure you save your output with an appropriate name for later use (this will be explained below). Once the output is ready you can use it like any other image inside any document, .pages, .docx, .tex and others.

Plotting functions

Now that you have gnuplot set up and are in the gnuplot environment it is time to draw your first plot. Type the following command and hit return/enter:

```
plot sin(x)
```

You should see the sine curve output on a separate window as shown in fig. 1 below. You can plot other functions as well, like `cos(x)`, `atan(x)` etc. using the `plot` ⟨`function`⟩ command. The function you plot is of the form $y = f(x)$ where

the `plot` command takes $f(x)$ as its argument, which is why $y = \sin(x)$ is plotted as `plot sin(x)` and $y = x^2$ is plotted as `plot x^2` and so on.

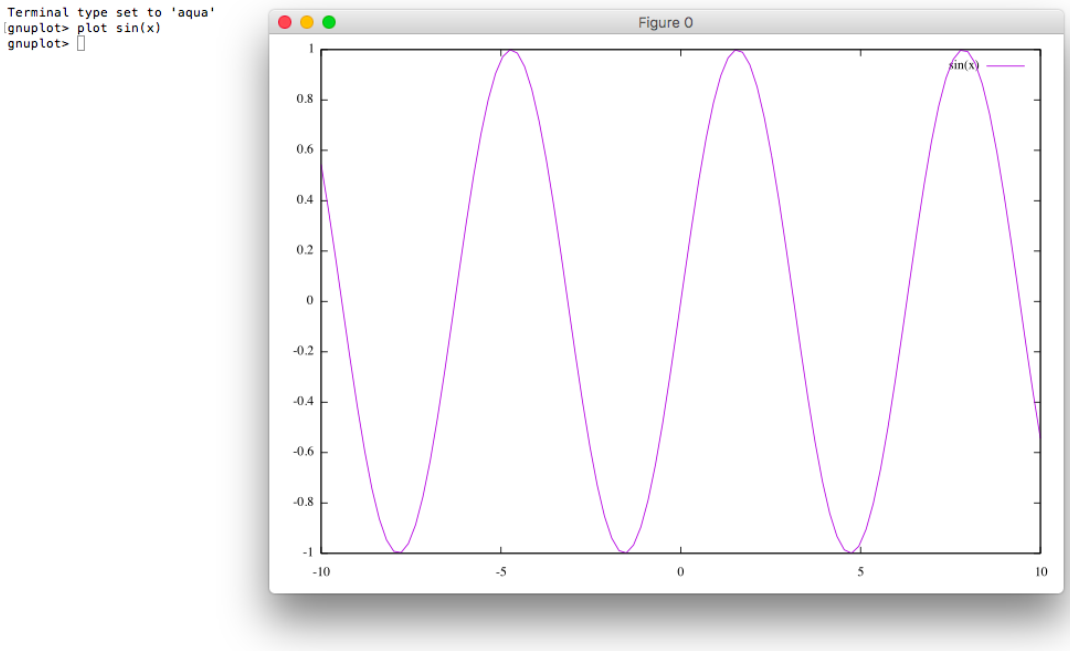


Figure 1: Plotting $\sin(x)$.

Let us now plot the trajectory of a projectile and use that graph as our main object to learn other gnuplot commands. Use the following command:

```
plot tan(pi/4)*x-((9.8/2*10*10*(cos(pi/4))**2)*x**2)
```

Observe that this is our standard projectile trajectory given by the equation

$$y = \tan(\theta)x - \frac{gx^2}{2u^2\cos^2(\theta)}$$

where we have used a $45°$ angle of launch θ, in radian, and an initial velocity of 10 units. The use of two asterisks, as in `x**2` implies indices, in this case x squared. We have assumed the magnitude of the acceleration due to gravity is $g = 9.8\,\mathrm{ms}^{-2}$ in our example. The `plot` function takes as its argument only the right-hand $f(x)$ side of the $y = f(x)$ equation above. As expected we end up with a parabolic trajectory like the output shown in fig. 2.

Refining outputs

Of course there are a few problems with our output in terms of its presentation. The graph is perfect, but the legend/key, the axes etc. can use some improvements. This is what the remainder of this section will focus on.

First let us start by getting rid of that equation at the top blocking our graph. This is particularly useful if you have multiple lines as it serves as a legend or key. In gnuplot commands must be given one after the other, often in any order, but

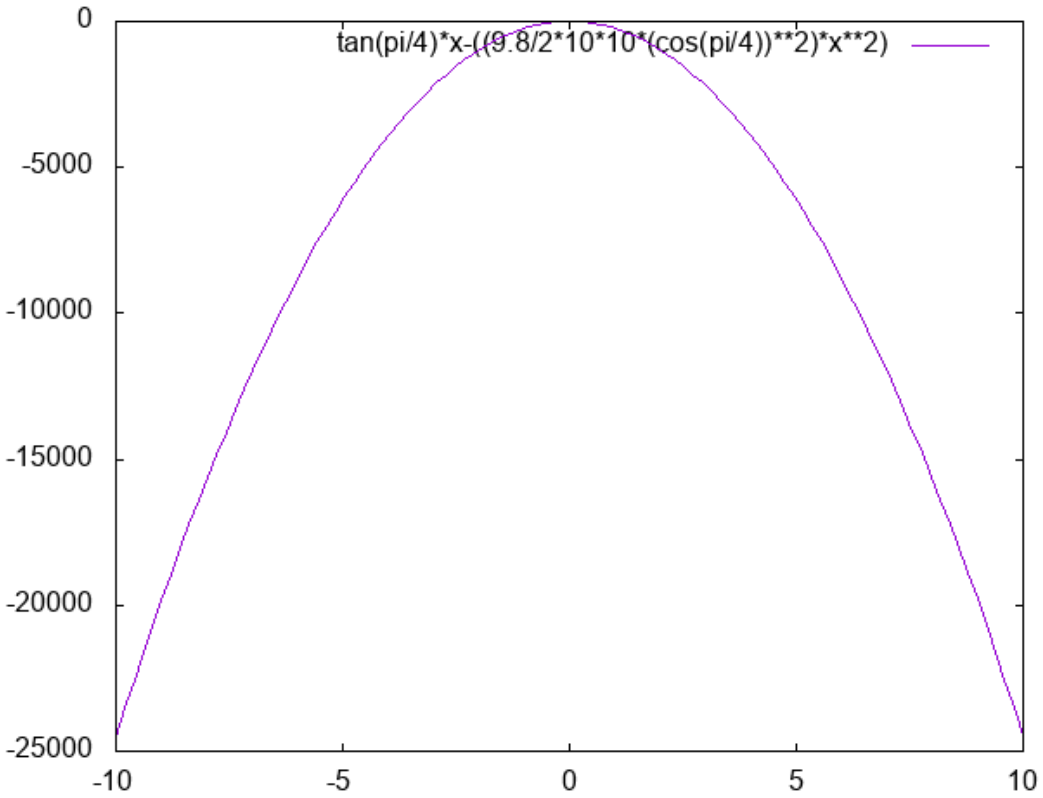


Figure 2: Projectile trajectory.

before the plot (or equivalent) command is given. gnuplot keeps these *settings* in memory for the current and subsequent plots.
To remove the legend use the command

```
unset key
```

or you can move the legend around to a more convenient location by specifying that location, such as the top right or bottom centre:

```
set key top right
```

```
set key bottom center
```

Most such commands are given using the `set` keyword in gnuplot so they are easy to recognise. Likewise `unset` undoes a setting. To reset all settings simply type

```
reset
```

Let us go a few steps further and label our axes, title our graph, draw a grid for better coördinates and so on. In gnuplot type the following commands and hitting return/enter after each. Nothing will happen visibly when you do this as gnuplot is simply committing these new settings to memory. When you finally plot your data

or function you can see the result of these settings.

```
set xlabel "Distance (m)"
```

```
set ylabel "Height (m)"
```

```
set title "Projectile"
```

```
set zeroaxis
```

```
set grid
```

These five commands should be self-explanatory. They set the x-axis label, the y-axis label, the graph title, display the (0,0) axes and display a graph/grid in the background respectively. Once you do this run the `plot` the command again for our trajectory equation and you should see an output that looks like fig. 3 below.

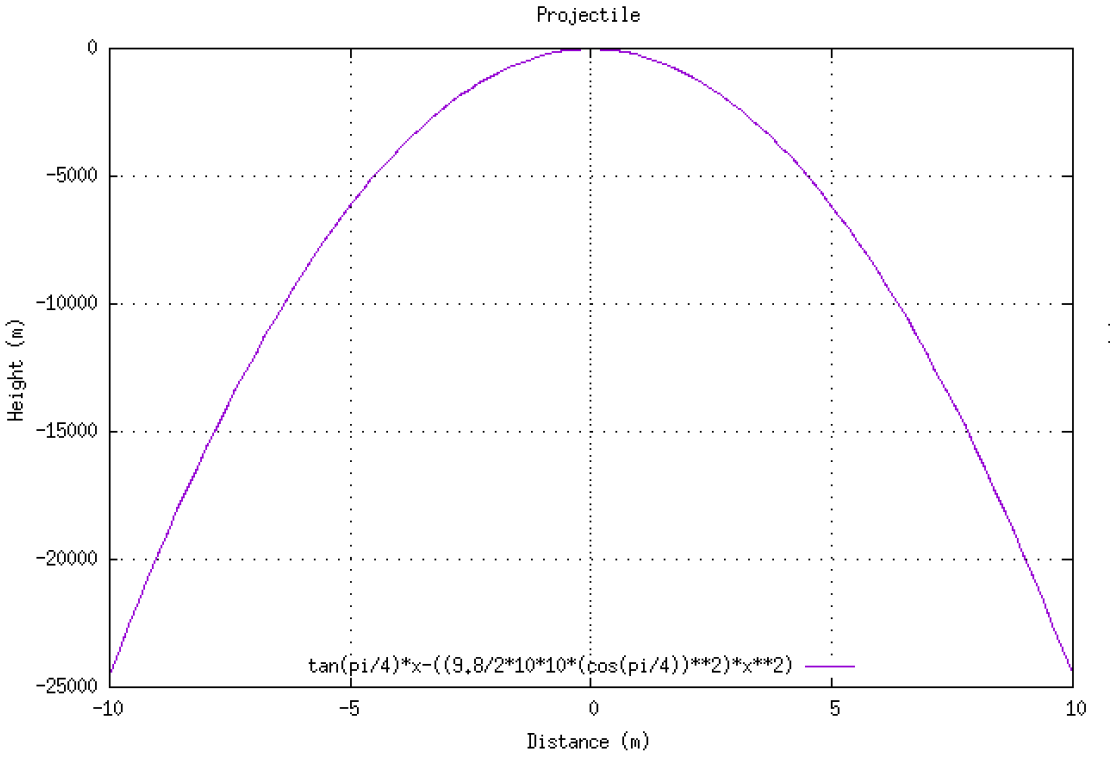


Figure 3: A better-looking projectile trajectory.

This is much better looking. The key is not obstructing our view, there is a nice title and the axes are now labelled. But the exact numbers on the axis are disorienting (to me anyway) so let us try to improve the precise numbering of axes.

First we set new ranges for our axes with the `[minimum:maximum]` syntax:

```
set yrange [0:30000]
```

```
set xrange [-10:10]
```

Notice that our graph starts at $-25,000$ m so, as our final step, we need to offset the entire graph up by that value to bring it to 0 m which we do by simply adding 25000 to $f(x)$ as follows:

```
plot (tan(pi/4)*x-((9.8/2*10*10*(cos(pi/4))**2)*x**2))+25000
```

So far we have made some good improvements but the graph still does not let us quickly check x and y values. Having more gridlines would help us; in other words, we need to reduce the least count of our axes to, perhaps, 2 m on the x- and 2,500 m on the y-axis. You should now see a more comprehensible output as in fig. 4 below.

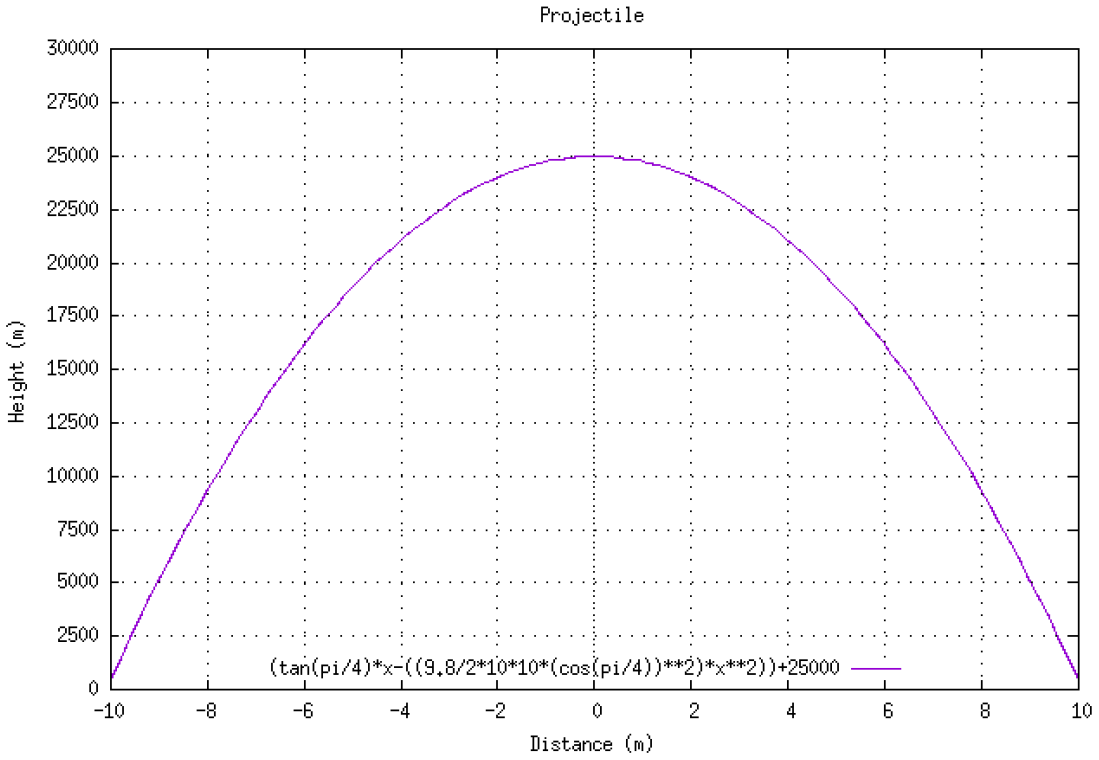


Figure 4: Improved ranges.

You can even switch scales from decimal (used here) to logarithmic with

```
set logscale x
```

```
set logscale y
```

for the x- and y-axis respectively. You can even add minor gridlines in-between existing ones, say for every 0.5 m on the x- and 500 m on the y-axis in the graph above, using the commands

```
set mxtics 0.5
```

```
set mytics 500
```

To quickly plot your last plot, instead of retyping the `plot` command, say

```
replot
```

You can even plot multiple plots on the same graph. To have some fun let us see the effect of the launch angle on our projectile. To see the effect of launch angles $\pi/3$ and $\pi/6$ on our projectiles we replace the angles appropriately in our equation and then use the command to plot multiple functions by simply separating each function with a comma as follows:

```
plot (tan(pi/4)*x-((9.8/2*10*10*(cos(pi/4))**2)*x**2))+25000,
(tan(pi/3)*x-((9.8/2*10*10*(cos(pi/3))**2)*x**2))+25000,
(tan(pi/6)*x-((9.8/2*10*10*(cos(pi/6))**2)*x**2))+25000
```

Let us also refine our axes and legend with the following commands:

```
set xrange [0:15]
```

```
set yrange [0:27000]
```

```
set key top right
```

and then `replot` to get the output shown in fig. 5.

Plotting from files

A lot of experimental set-ups output data as large sets of numbers. Such ordered pairs can also be plot using gnuplot by simply mentioning the file name. Say we have a file called `data.dat` with the contents shown below. The data is simply separated by tabbed spaces and each visible column is interpreted as just that by gnuplot so the spacing is all we need to pay attention to. You can use any text editor, from TextEdit to VI to Notepad, to create such a data file.

Say we want to plot the first and second column against each other, we can do so using the following command:

```
plot "data.dat" u 1:2 w linespoints
```

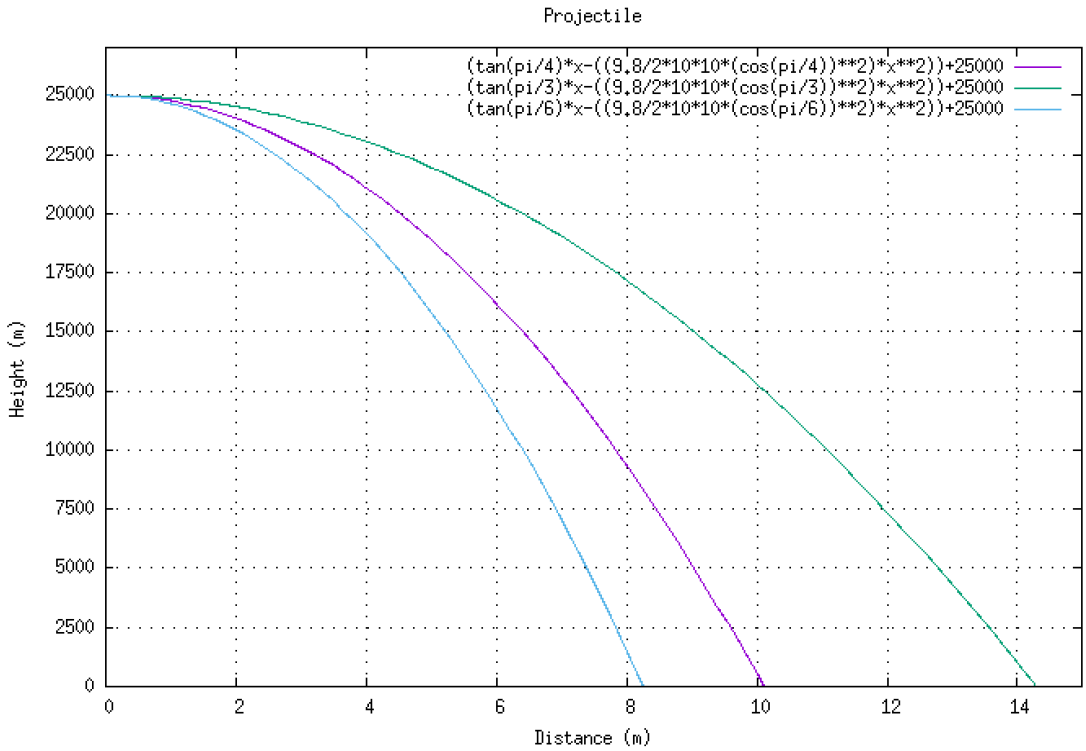


Figure 5: Multiple plots in one graph.

Remember to first `reset` your gnuplot or x- and y-axis ranges from the previous plot will be applied to this as well. You can alternately exit and return to gnuplot but that is a little time consuming compared to the good old `reset` command.

0	0	4
2	3	5
4	4	7
6	5.75	11
8	9	16.25
10	10.5	22

`data.dat`

If you did it right you should have a plot that looks like fig. 6. There are two new keywords in this `plot` command: the `u` stands for 'using' and the `w` stands for 'with' and the `1:2` tells gnuplot to plot the first column as the x-axis and the second as the y-axis. You can also use the full keywords `using` and `with` instead of their respective single-letter shorthands.

Also the keyword `linespoints` tells gnuplot to mark coördinates with points and connect them with lines. You can just as well choose to have only `points` or `lines` on your graph.

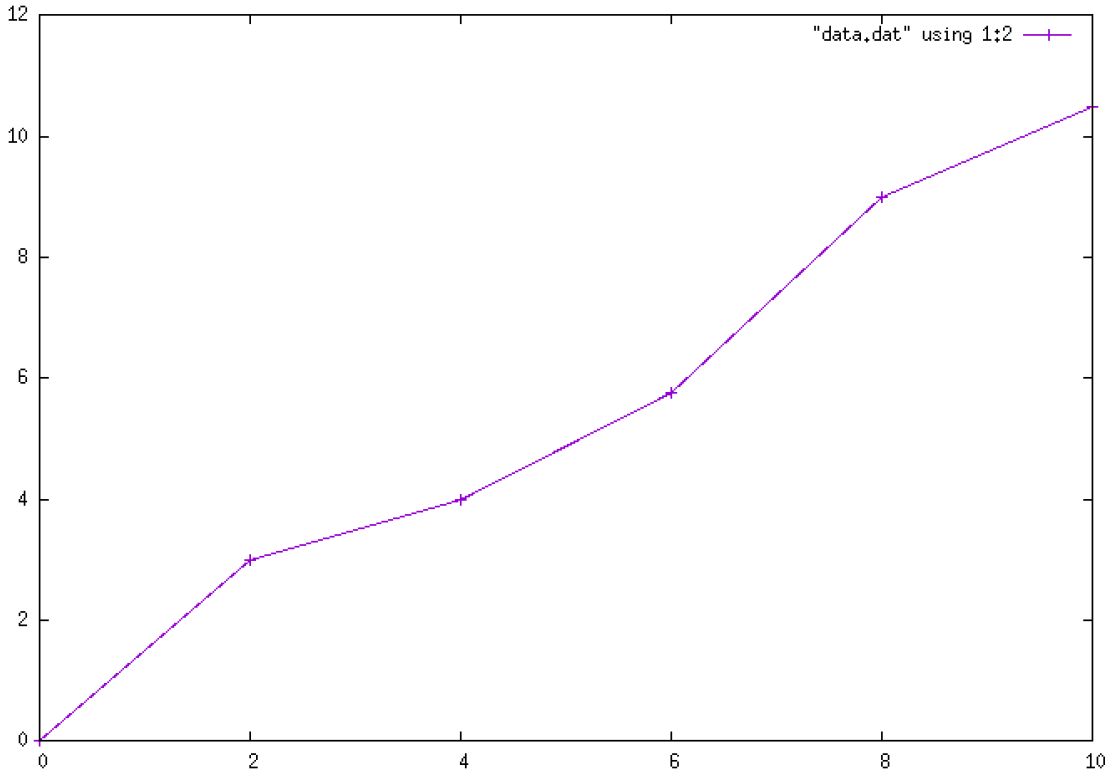


Figure 6: Plotting from a file.

Fitting data

One observation to make is that fig. 6 is *not* a straight-line graph. What if we wanted to take this data and fit it into a straight line? That too can be done in gnuplot with a couple of simple commands. (The letter `t` is short for title; note how the title is used in the key when plotting multiple plots on one graph.)

```
f(x) = m * x + c
```

```
fit f(x) "data.dat" u 1:2 via m,c
```

```
plot "data.dat" w points, m * x + c w lines t "Line-fit curve"
```

After you execute the `fit` command gnuplot shows you its fitting calculations and the values it has arrived at for the slope (m) and intercept (c). This is what allows us to use `m * x  c+` in the next command to plot a straight-line into which our data fits. We can also fit our data into cubic splines and Bezier curves with:

```
plot "data.dat" u 1:3 smooth csplines title "Cubic spline",
"data.dat" u 1:3 w lines title "Raw",
"data.dat" u 1:3 smooth sbezier title "Bezier"
```

to arrive at the multi-plot in fig. 8. Note the subtle differences between the curves.

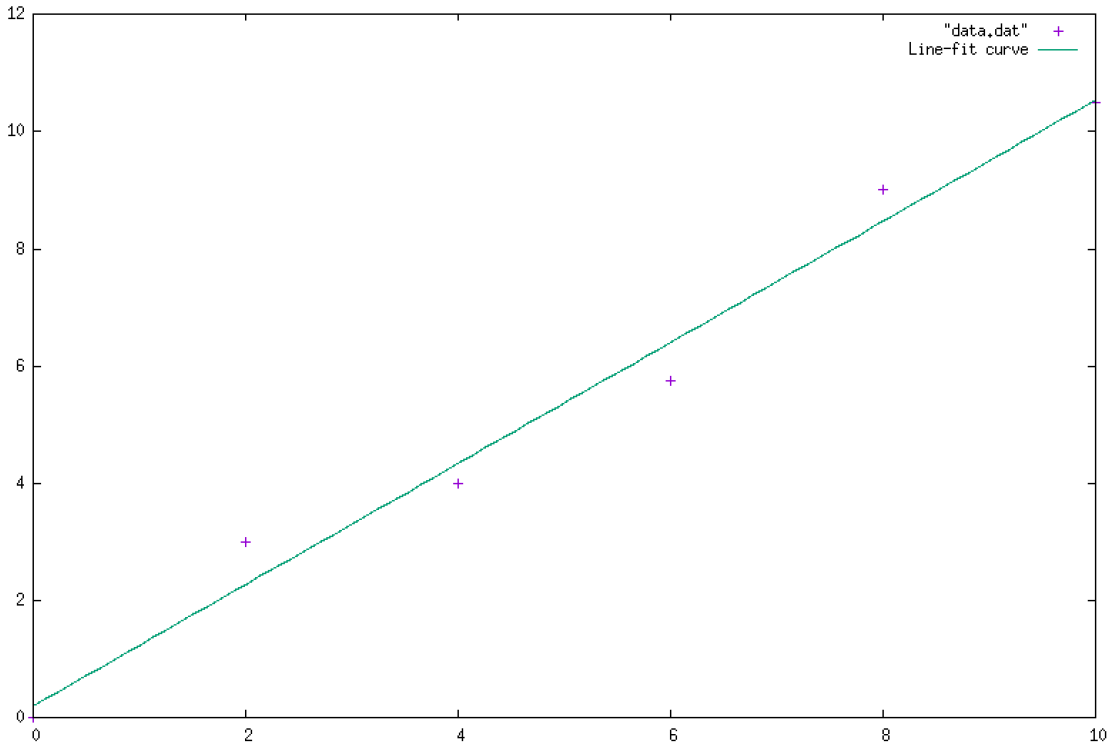


Figure 7: Fitting data to a line.

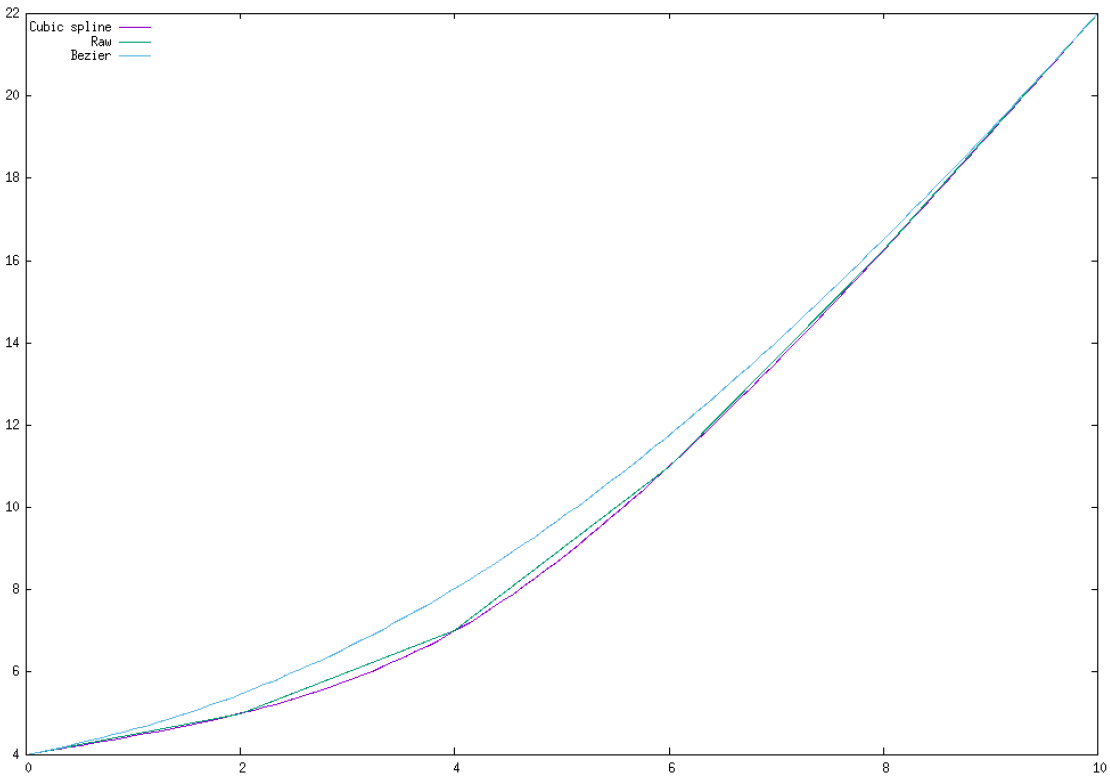


Figure 8: From left to right: Bezier, raw and cubic spline fits.

When you are typing a command in gnuplot if you find the line becoming longer than you like, use the backward slash '\' character: type your command, hit \, hit return/enter and continue typing. gnuplot interprets such lines as one command.

Saving files

The clever reader may have noticed by now that none of their carefully drawn plots are being saved anywhere. This is by design. You can use gnuplot to keep refining your plot as much as you want and when you are finally ready to save your finalised work you can give it a name and save it:

```
set output "mygraph.eps"
```

Now plot your file or function and it will be saved into your current working directory as an .eps file[2]. This only works if, as we said at the start of this guide, you used the `postscript eps enhanced` terminal.

Alternately you can use the `set terminal png` or the `set terminal gif` commands to enable .png and .gif outputs respectively. Also try the `latex` terminal to get a `picture` environment code as your output which you can then simply copy and paste into a LaTeX file[3].

Be sure to enter your working directory *before* you start gnuplot because all your outputs will start getting saved into that directory. Exiting gnuplot in-between will reset all your old plot settings.

This guide should have whet your appetite about gnuplot. As stated in the beginning, this is not an exhaustive guide but you should, by now, be in a position to explore gnuplot all by yourself and discover several other capabilities. Start by looking at, for example, colouring your graphs, and drawing three-dimensional plots. And remember to always have fun.

* * *

Ever since the first edition of this self-study guide was published several readers wrote to me enquiring about the code that would output a torus like the one on the cover page. Here it is for you to try:

```
set nokey
set parametric
set hidden3d
set view 30
set isosamples 30,20
splot [-pi:pi][-pi:pi] cos(u)*(cos(v)+3), sin(u)*(cos(v)+3), sin(v)
```

[2] Encapsulated PostScript.

[3] Such a `picture` environment code, while incredibly convenient, can become extremely lengthy. For instance, plotting fig. 8 as a LaTeX `picture` leaves you with 475 lines of code for a single plot.

Notes