



Visualize your data with gnuplot

Advanced graphs and data-plotting mastery at your disposal

Level: Introductory

Nishanth Sastry (nishanth_sastry@us.ibm.com), Staff Software Engineer, IBM

22 Jul 2004

Turn your data and functions into professional-looking graphs with Gnuplot 4.0, a freely distributed plotting tool. In this article, get a hands-on guide to gnuplot that emphasizes the idioms you'll need to use this tool effectively.

Gnuplot is a freely distributed plotting tool with ports available for nearly every major platform. It can be operated in one of two modes: when you need to adjust and prettify a graph to "get it just right," you can operate it in interactive mode by issuing commands at the gnuplot prompt. Alternately, gnuplot can read commands from a file and produce graphs in batch mode. Batch-mode capability is especially useful if you are running a series of experiments and need to view graphs of the results after each run, for example; or when you need to return to a graph to modify some aspect long after the graph was originally generated. While it is hard to capture the mouse-clicks that you would use to prettify a graph in a WYSIWIG editor, you can easily save gnuplot commands in a file and load them up to re-execute in an interactive session six months later.

Gnuplot was originally developed in 1986 by Colin Kelley and Thomas Williams. A number of contributors added support for different "terminals," creating variants. In 1989 and 1990, these were merged into gnuplot 2.0. And in April 2004, version 4.0 was announced. This tutorial will apply to version 4.0, but most of the commands introduced here should apply to other versions as well. Where possible, I will mention the major differences. The official Web site for gnuplot is listed in [Resources](#).

In the following, we present a hands-on tutorial for the novice; but even if you have prior experience with gnuplot, you might find new idioms and new commands introduced in version 4.0. We start with a simple sine curve and customize it to look just as we want it to. Then we will study how to plot a set of data points. In this article, we confine ourselves to 2D plots, since they are the most common.

GNG: Gnuplot's Not GNU

While its name may imply otherwise, gnuplot is not covered by the GPL. For the legally curious, gnuplot FAQ #1.7 says: "Gnuplot is freeware in the sense that you don't have to pay for it. However, it is not freeware in the sense that you would be allowed to distribute a modified version of your gnuplot freely. Please read and accept the Copyright file in your distribution."

Oftentimes, novice users may have a pretty good idea about how their graphs should look, but it is not apparent what gnuplot command they need to use. So the key to understanding gnuplot is to get a good grasp of its vocabulary, and the rest should be intuitive enough. In this tutorial, I will only be able to provide a feel for some of the common options you will end up using in gnuplot; it is by no means exhaustive. So, for example, I tell you how to set the x-range to limit the range on the x-axis of the graph. Setting the y-range is similar (use `yrange` instead of `xrange` in the command), but I don't talk about it.

The basics

Fire up gnuplot by typing `gnuplot` at the command prompt of your shell. The first thing you will see is the prompt sign `>`. The prompt is the point of input into gnuplot; Linux users will be used to its behavior. For example, you can use arrow keys to navigate the history of commands you have typed before, then edit and re-execute them; and the

Home and End keys work as expected. gnuplot can be recompiled to use the GNU readline library to move around on the input prompt, but the defaults function similarly.

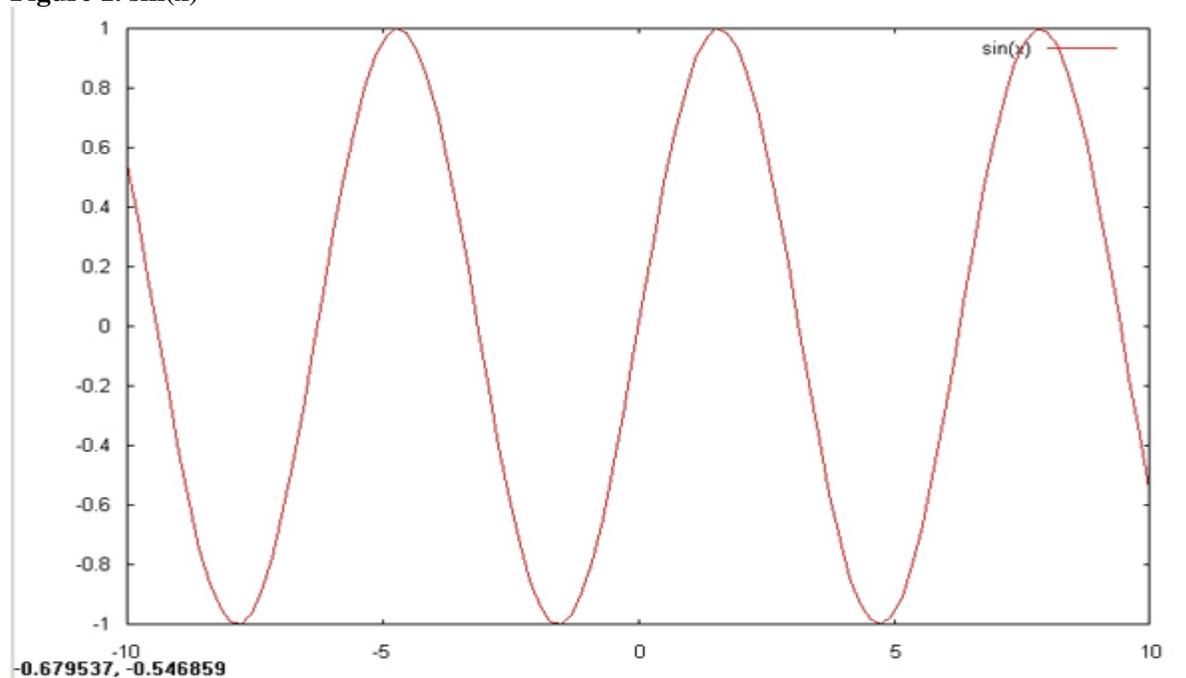
Gnuplot comes with extensive online help, which you cannot avoid using if you intend to do anything non-trivial. The syntax is uniform: help for any command is obtained by typing `help <command>`. Go ahead and start up gnuplot, and try the commands `help set yrange` and `help set now` (use `q` to quit the help after each command). Notice that `yrange` is one of the sub-options available under `help set`. In general, gnuplot help offers further help for all possible customizations of a command. Glancing at the examples section in the help is usually more than sufficient to understand how to use a command.

Gnuplot also has an extensive set of demos that showcase its capabilities, usually located in the `demo` subdirectory of the install. To get the tour-de-force, change into this directory at the gnuplot prompt (for example, `cd '/opt/gnuplot/demo'` -- note that gnuplot requires all file and directory names to be enclosed in single or double quotes) and type in `load 'all.dem'`. Individual `.dem` files in this directory demonstrate individual functions, and `all.dem` loads them all one at a time. You may want to hold off on this exercise until the end of this article, though, so that we can get started without further ado...

For those who want to follow along, you may type each line of code in the code listings individually at the prompt. Alternatively, save the whole listing into a temporary file and run it by typing `load 'filename'` at the gnuplot prompt (don't forget the quotes).

The command for 2D plots is, predictably, `plot`. Type in `plot sin(x)` at the prompt: you should see the familiar sine curve in a pop-up window.

Figure 1. $\sin(x)$



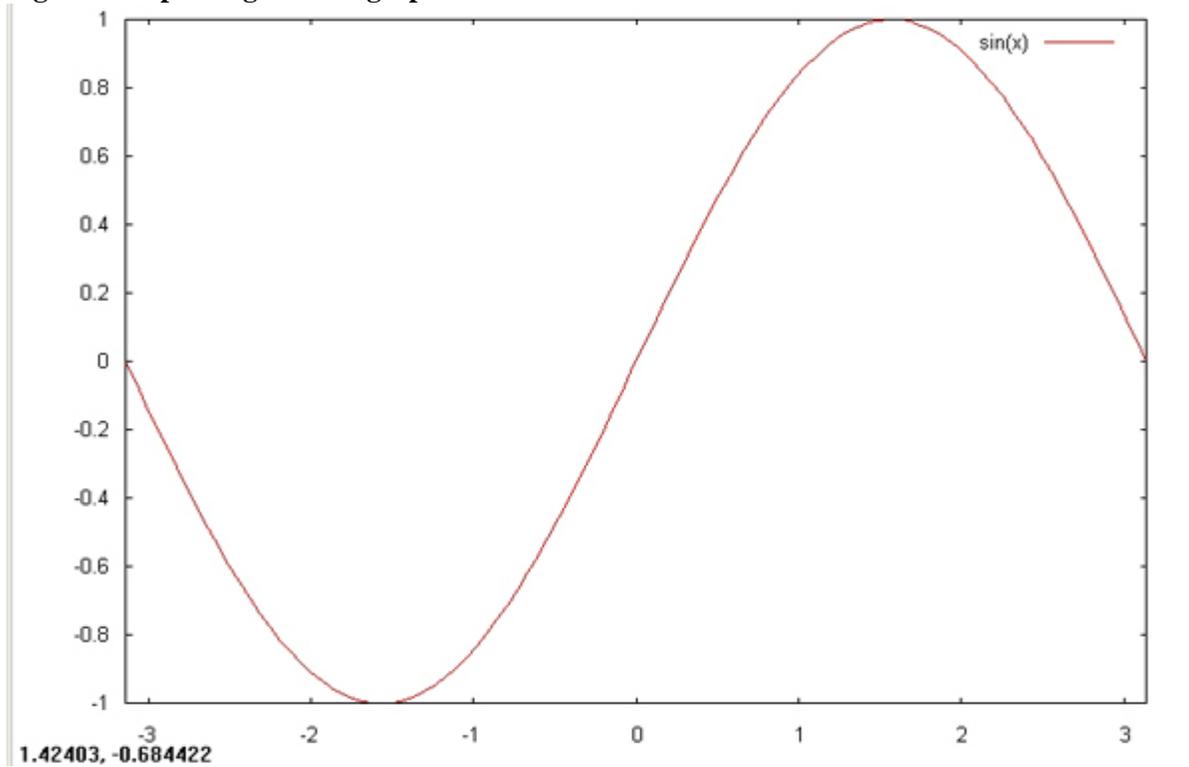
We just created the simplest possible graph. Let us see how we can customize it in different ways. Let us say we want to see just one cycle of the sine wave. We do this by restricting the default x-range of the graph. Ranges are specified using the notation `[min:max]`. To specify only the min, use `[min:]` and to specify only the max, use `[:max]`. Mathematicians know this as the notation for so-called "closed" intervals.

Here we use `[-pi:pi]` to get one sinusoid cycle:

Listing 1. $\sin(x)$ from $-\pi$ to $+\pi$

```
set xrange [-pi:pi]
replot
```

```
reset
```

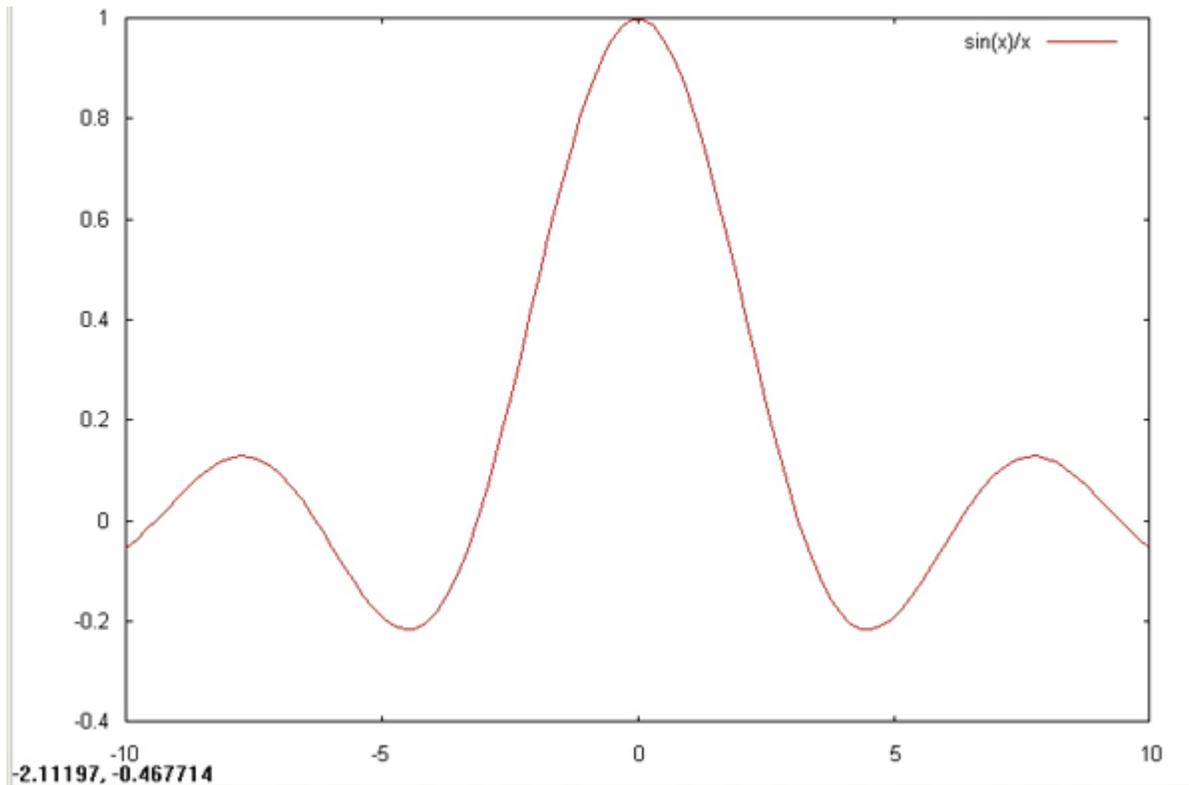
Figure 2. Replotting the sine graph

We just used the `replot` command, which executes the previous plot command. This can be very useful when you are composing a graph and need to keep changing it to add desired features. Additionally, `replot` lets you add more graphs. Try typing in `replot cos(x)`. Syntactically, this is equivalent to `plot sin(x), cos(x)`. `Replot` simply takes the previous plot string, adds an implied comma, and then appends the rest of the input given to it.

The last command used, `reset`, is strictly not necessary for plotting the sine curve. It removes the effect of all previous `set` commands and restores default values. Thus, in this case, it will use the default x-range.

Now let's try something more fancy: `plot sin(x)/x`. You should see something like this:

Figure 3. $\sin(x)/x$



From just these three simple examples, you can see that gnuplot understands pi and has a rich and natural vocabulary of mathematical functions. It even knows useful statistical functions governing normal distribution, and esoteric special functions like the lambert, bessel, beta and gamma functions (and more!) that you would ordinarily only expect from a full-fledged math tool like mathematica. A rule of thumb is that the syntax is similar to the syntax in C, which in turn is similar to the syntax used in everyday mathematics. (An important exception is the notation for exponentiation: x raised to the power y is conveniently written as x^*y).

Next, let us give titles to the graph ("My First Graph") and the axes (x-axis is "angle, in degrees," and y-axis is "sin(angle)").

Listing 2. Giving titles to the graph and axes

```
set title "My first graph"
set xlabel "Angle, \n in degrees"
set ylabel "sin(angle)"
plot sin(x)
```

Notice that the `\n` in the xlabel produces a new line. In general, gnuplot does C-like backslash processing within double quoted strings. Windows users beware: If you tend to use double quoted strings for filenames, you will have to use two backslashes: for example, "c:\\developerworks" (or you can use a forward slash: "c:/developerworks").

Now, we notice that the x-axis is not really marked in degrees and does not look good. Let us fix this by adjusting the tic marks on the x-axis to make named (major) marks only at 90 degree increments and minor marks at 45 degree increments. The major tics have a "level" of 0, which is the default; and the minor tics have a level of 1. Each point is separately specified by a 3-tuple: "label" (in quotes), <point-at-which-tic-is-made>, and <optional-level>.

Listing 3. Altering the tics on the axes and setting a grid

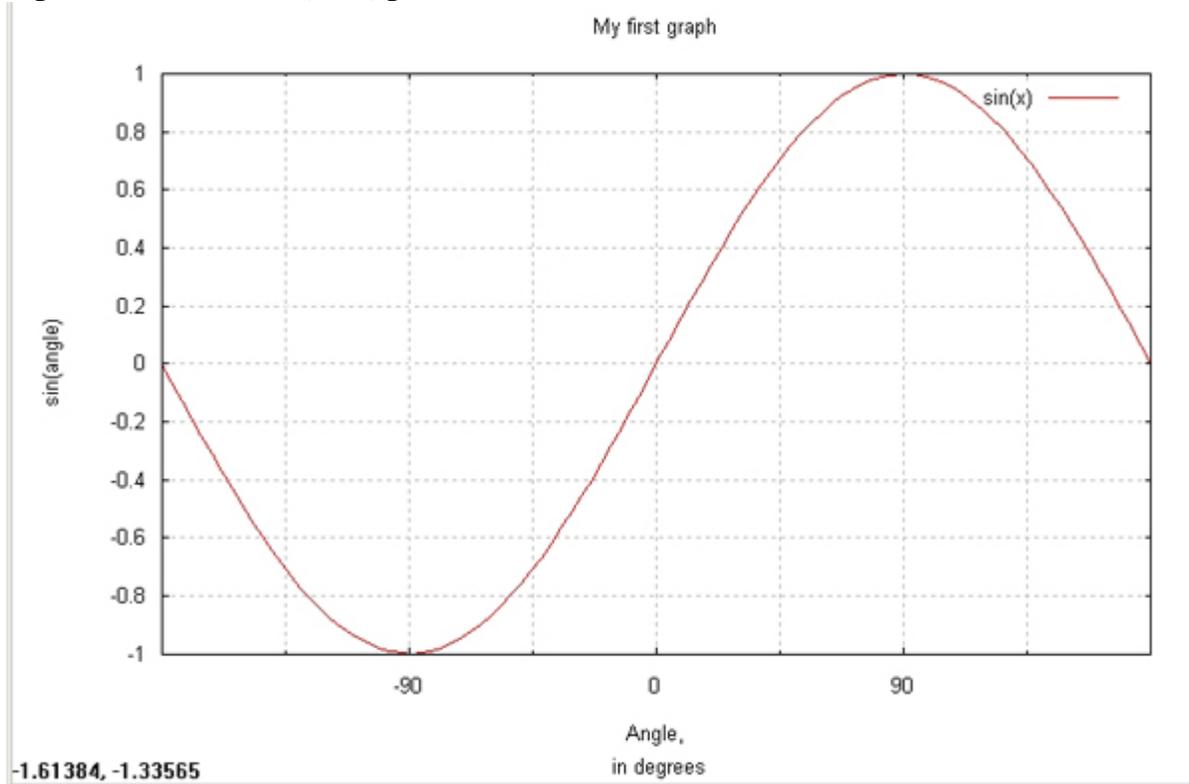
```
set xrange [-pi:pi] # we want only one cycle
set xtics ("0" 0, \
           "90" pi/2, "-90" -pi/2, \
           "" pi/4 1, "" -pi/4 1, \
```

```

                                "" 3*pi/4 1, "" -3*pi/4 1)
set grid
set xlabel "Angle, \n in degrees"
set ylabel "sin(angle)"
plot sin(x)

```

Figure 4. $\sin(x)$ with tics, title, grid and labeled axes



Notice the use of empty labels ("") for the minor marks at +45 and -45 degrees. Also observe that we don't need to list the points in any order. Like everything else in gnuplot, the `xtics` command is enormously versatile and there are convenient options for creating tic marks at regular increments (regular multiples, in logarithmic scale), and special formats for time-related data. Look up the examples in `help set xtics` for the juicy stuff.

As a side note, observe that we use a backslash to extend the `xtics` command over more than one line of input. This can make a long or complex command more readable.

We also slapped on a grid with `set grid`, which lets you eye the graph quantitatively. We have been extensively using the `set` command until now. Gnuplot has a consistent way to undo sets: in gnuplot 4.0, the command is `unset`. For example, if you don't like grids, use `unset grid`. Prior to 4.0, the command would have been `set nogrid`.

Incidentally, all the examples we give can be run in interactive mode or in batch mode. To run it in interactive mode, type each command at the prompt. To run it in batch mode, type or cat the commands into a text file and either read it in at the gnuplot prompt using `load 'filename'`, or give it as an argument to gnuplot at the shell prompt: `$ gnuplot filename` With the second method, gnuplot exits after executing all the commands in the input file, so it is especially efficient for batch-mode execution when plotting directly to a file (see next section). If the second method is used when plotting onscreen, the output window will disappear as soon as it is rendered (when gnuplot exits), so you will need to use an explicit "pause -1" command (see `help pause`) to make the window stick.

Gnuplot command files used in batch-mode operation tend to stick around for a while, so it helps to use comments to improve readability. Anything after a `#` until the end of line is a comment. So in the example in Listing 3, gnuplot does not process we want only one cycle.

Terminals -- a digression

On startup, you may have noticed that the terminal type is set to `x11`. Gnuplot has a modular design and can plot on several terminal devices. This includes printing directly onto several kinds of printers including Epson, HP and Imagen printers. It can even plot on pseudo-devices such as `postscript` and `png`. Essentially, this involves producing an output file instead of a viewable file or a printout. This is the trick to including your graphs within other reports. Although I have not noticed any difference on Windows and Linux, the documentation recommends that you set the terminal type before setting the output file name:

Listing 4. setting output and terminal type

```
set terminal png          # gnuplot recommends setting terminal before output
set output "output.png" # The output filename; to be set after setting
                        # terminal
replot
```

Now, the file `output.png` has the graph you just plotted and can be included in a larger report. There are specialized terminals for many popular typesetting and drawing programs, including Adobe Illustrator (`set term aifm`), Corel Draw (`set term corel`), AutoCad (`set term dxf`), and several LaTeX-related terminals (`eepic`, `latex`, `pstricks`, `texdraw`, `tpic`, and so on). You can also get files in PostScript, enhanced PostScript (`eps`) or the Adobe PDF formats. Your version of gnuplot may not be compiled with all the terminal drivers -- for example, Windows users do not need the `X11` terminal, and Linux users don't need the Windows terminal. Similarly, due to legal reasons, gnuplot 4.0 removed support for gif terminals.

All terminals are not alike in their capabilities. Some, like LaTeX, may not support rotation of text, so if you set the `ylabel`, as we did earlier, it would appear differently on different terminals. On the other hand, you can use LaTeX commands in LaTeX-specific terminals. For example, `set ylabel "$\sin(\theta)$"` (notice we use two backslashes to produce the single backslash required by LaTeX -- gnuplot does the backslash processing on double-quoted strings before passing it to the terminal drivers). Now, you can include the output file in your LaTeX file using `\input{output.tex}`. To achieve the same in PostScript terminals, use the corresponding PostScript command: `{/Symbol q}`. In both enhanced PostScript and LaTeX, you can get superscripts using the notation `x^{superscript}` and `x_{subscript}`. Notice also the abbreviated terminal and output commands:

Listing 5. Capabilities of the eps driver

```
set term post enh          # enhanced PostScript, essentially PostScript
                          # with bounding boxes
set out 'gplt.eps'

set xlabel '{/Symbol q_1}'
set ylabel 'sin^2({/Symbol q_1})'
plot sin(x)**2
```

The code supplement in the [Resources](#) section has an example TeX file that showcases the above commands.

Mouse support

New to version 4.0, the Windows and X11 terminals support mouse interaction. Mouse support has two main uses: coordinate tracking, and zooming. The current coordinates of the mouse pointer are shown on the lower left corner of the terminal window. This can be copied to the clipboard by double clicking the first mouse button. This is useful when you need to set a label or an arrow which need explicit coordinates. (Use `help arrow` and `help label` to get more details on these commands.) You can also drag the second (or right-click) mouse button to select a rectangular area to zoom in on. In 3D plots, you can drag the mouse to rotate the image (pressing control changes only the view of the axes instead; so for graphs that take too long to render, you can change the axes first, and then release the control key to re-render the graph).

Mouse support also enables a bunch of useful hotkeys: **u** unzooms the graph if you have zoomed in previously. **g** toggles the grid, **l** toggles logscale on both axes; **L** toggles the logscale on the axis nearest to the pointer, **r** toggles a ruler, which establishes an arbitrary point of origin. When the ruler is enabled, the bottom of the screen shows the x and y distances of the current mouse pointer from ruler origin, as well as the x-y distances from the actual origin -- which is the same as the coordinates of the point). In 3D, the arrow keys can be used in lieu of the mouse drag to rotate the image. The space bar raises the command window, and **q** quits the terminal window. To see all the options, type **h** on a mouse-enabled terminal.

Controlling scale and aspect ratio

By default, gnuplot uses a scaling factor of 1 for both the x and y-axes, but it makes no effort to control the aspect ratio (the ratio of the length of the y-axis to the length of the x-axis) of the figure. The terminal driver uses the default aspect ratio of the terminal. Either the scaling factor or aspect ratio, or both, may be specified with the `set size` command, for example:

```
# square is synonymous to an aspect ratio of 1;
# scale y-axis by 2, retain x-axis size
set size ratio square 1,2
```

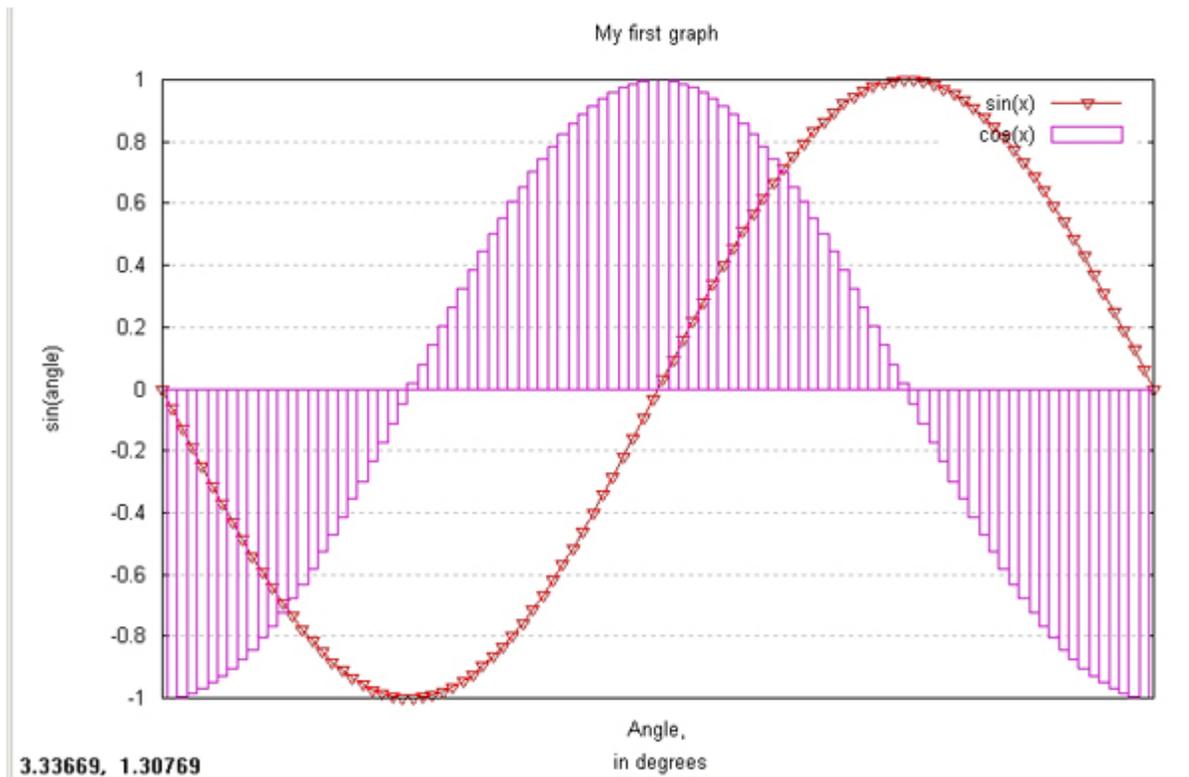
The success of gnuplot in drawing a graph with a given aspect ratio may be limited by the capabilities of the terminal. `set size` is also useful in conjunction with the `multiplot` command, which is used to produce more than one graph on the same output screen or file.

Plotting more than one curve

As the astute reader may have guessed from the comments above on `replot`, gnuplot lets you graph multiple lines simultaneously. Let us plot both the sine and the cosine curves. The simple plot command would be `plot sin(x), cos(x)`; the lines being graphed are separated by commas. If you do not specify anything else, gnuplot automatically displays the two graphs so that they are distinguishable from each other -- in terminals like Windows and X11, gnuplot uses different colors. Monochrome terminals display the graphs using different kinds of lines. By looking at the legend (or key), you should be able to tell which line in the graph is which. Alternatively, gnuplot lets you take greater control by specifying the style with which to draw a graph:

```
unset xtics # keep all other things simple
plot sin(x) with linespoints pointtype 5, cos(x) w boxes lt 4
```

Figure 5. Multiple curves



The `with` clause (see `help plot with`, and also `help plotting styles` if using version 4.0) lets you specify in dizzying detail, exactly how you want your lines to look. In this example, we show two useful styles. The first style, `linespoints`, is generally useful when plotting data. It marks a point at each sample or data point considered, and connects consecutive points using linear interpolation. Here we additionally specify the point type as `5`, which chooses the fifth kind of point that the terminal is capable of. The second style, `boxes`, is more appropriate when plotting histogram data. Notice how we abbreviate `with` as `w` in the `cos(x)` curve. Similarly, `lt` is an abbreviation for `linetype`, and is another terminal-specific setting that chooses the fourth kind of line that the terminal can draw. Needless to say (what, you did not try `help plot with` yet?), you can use `pt` instead of the verbose `pointtype`. If you intend to use the same plotting style across several lines (either on a single plot command or over several plot commands), you can set the plotting style with a `set` command. In gnuplot 4.0, use `set style function linespoints`; prior versions use the syntax `set function style boxes`. To change the style used for plotting data sets as opposed to functions, use `set style data linespoints` (`set data style linespoints` in prior versions).

How gnuplot does what it does

As an interesting aside, this exercise reveals how gnuplot actually plots functions -- it takes several samples and plots the value at each sample point. Usually, it smoothly interpolates between these points; using the `boxes` style forces it to use a flat histogram bar instead, and the `linespoints` style marks each sample with a point. To see this more clearly, explicitly set the sampling rate to 10 (`set samples 10`) and replot. (Then set it back to the default of 100).

The `with` clause is the first example where we used something other than a `set` to customize a graph. A few other commonly-used options can also be set directly in the plot command. For example, you can set the `xrange` also: `plot [-pi:pi] sin(x)`. Use this sort of syntax for a quick-and-dirty one-time change to the default plotting options. To re-use across multiple graphs, or across multiple plot commands, use the `set` command.

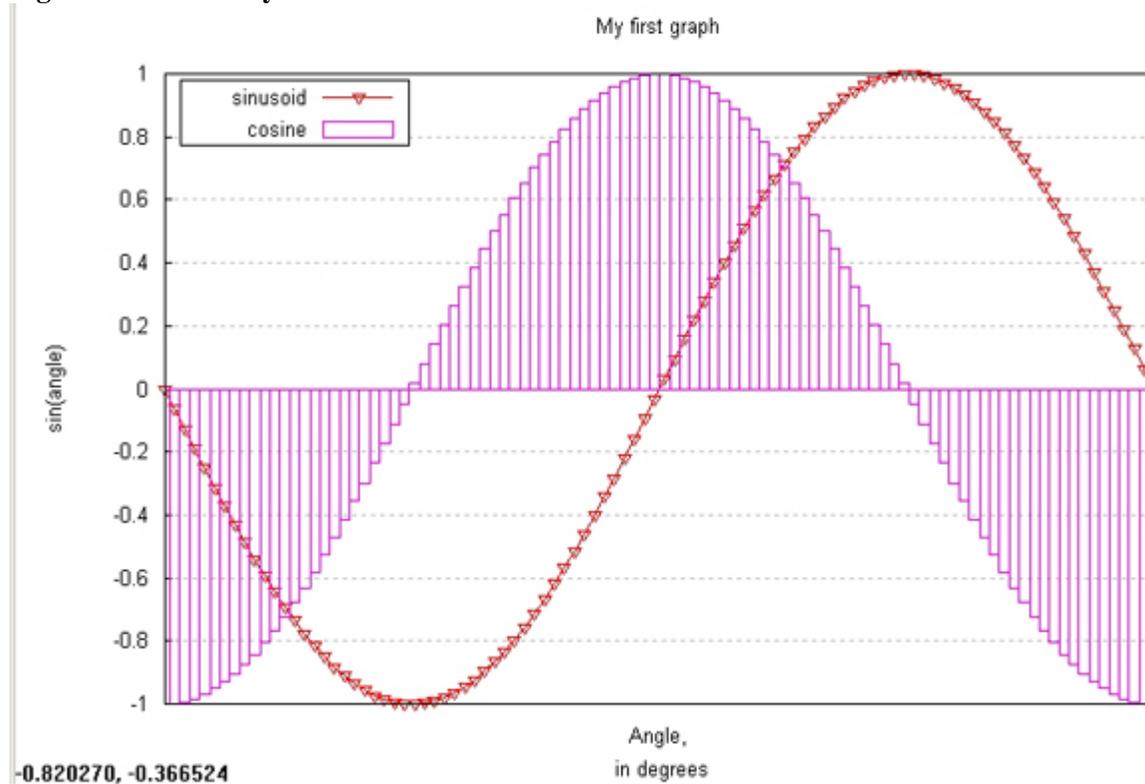
When plotting two or more curves, we need a key or legend to distinguish them. By default, the key is in the top right corner; but if it is interfering with your graph, you can place the key elsewhere -- even outside the graph, if you wish. The snippet below places the key on the top left corner, and sets a box around it. We also control the names given to curves in the legend by giving explicit titles in the plot command. The title can be abbreviated to `t`, as we do for the

cosine curve. Use `notitle` instead of `title` if you do not want a curve to be named in the key. A final point, and one that often trips up new users, is that the title must appear before the `with` clause, if any:

Listing 6. Customizing the key or legend of the graph

```
set key top left
set key box
plot [-pi:pi] sin(x) title "sinusoid" with linespoints pointtype 5, \
cos(x) t 'cosine' w boxes lt 4
```

Figure 6. Custom key



Plotting related graphs on the same screen

Above, we plotted both the sine and cosine on the same graph. Gnuplot also lets you plot more than one graph so that they appear side-by-side on the same output screen or file. On some typesetting systems, it is easier to include two related figures as one file than to include them separately.

Here is a simple annotated `multiplot` that plots the sine, cosine, and their respective reciprocals. The basic approach is to set a scale of 1 for the bounding boxes of the screen or image file. Then, we plot the different graphs so that they each take up one fourth of the screen. We also control where the graph appears by setting the origin using screen coordinates (see `help coordinates` for a discussion on the different kinds of coordinate systems). Also, note that each plot also automatically calculates widths for the axes, tics, and so on -- so you may have to tinker with the margin widths to get the axes of the different plots to line up. In this case, I had to set the left margin to 6-characters wide to accommodate the tics in scientific notation for $\operatorname{cosec}(x) = 1/\sin(x)$.

Listing 7. Multiplot example

```
set xrange [-pi:pi]

# Uncomment the following to line up the axes
# set lmargin 6

# Gnuplot recommends setting the size and origin before going to
# multiplot mode
```

```

# This sets up bounding boxes and may be required on some terminals
set size 1,1
set origin 0,0

# Done interactively, this takes gnuplot into multiplot mode
# and brings up a new prompt ("multiplot >" instead of "gnuplot >")
set multiplot

# plot the first graph so that it takes a quarter of the screen
set size 0.5,0.5
set origin 0,0.5
plot sin(x)

# plot the second graph so that it takes a quarter of the screen
set size 0.5,0.5
set origin 0,0
plot 1/sin(x)

# plot the third graph so that it takes a quarter of the screen
set size 0.5,0.5
set origin 0.5,0.5
plot cos(x)

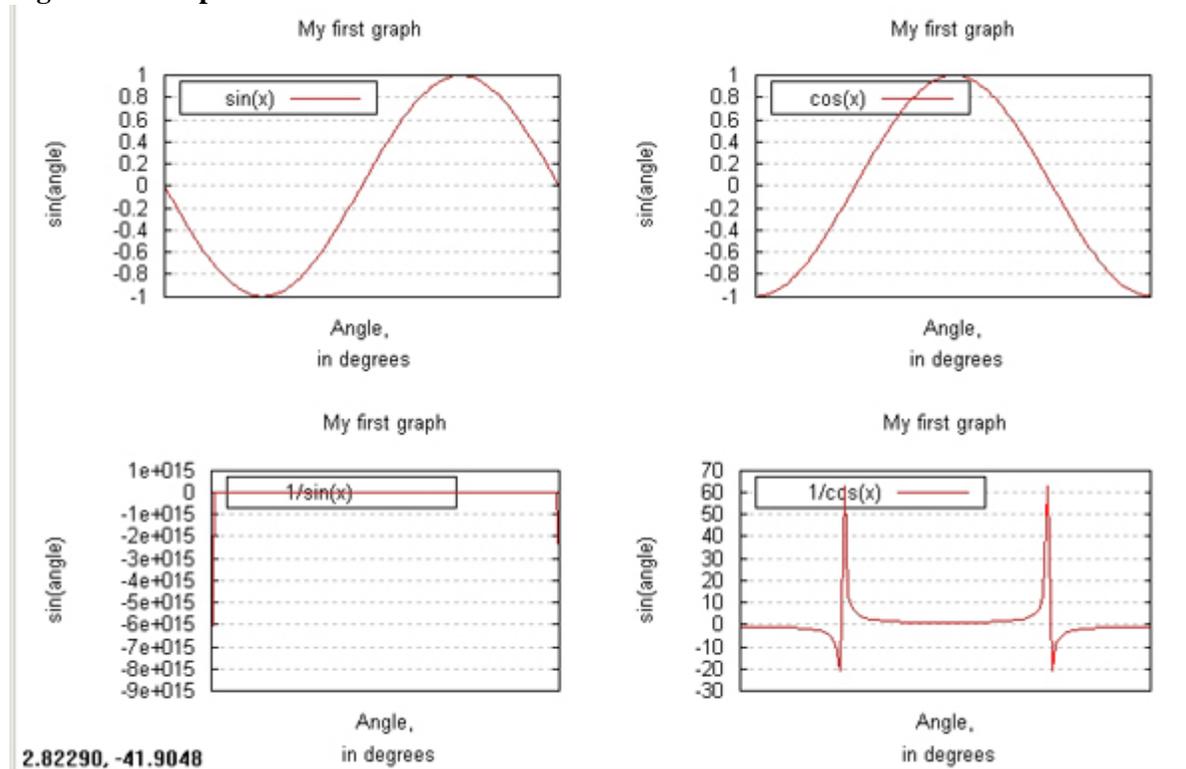
# plot the fourth graph so that it takes a quarter of the screen
set size 0.5,0.5
set origin 0.5,0
plot 1/cos(x)

# On some terminals, nothing gets plotted until this command is issued
unset multiplot

# remove all customization
reset

```

Figure 7. Multiplot madness



Plotting data

While most of this tutorial has, for expository purposes, concentrated on plotting the sinusoid, you will most likely be plotting data from experiments or sales data or the like. In this section, we will demonstrate how to plot different graphs using IBM's stock price as the data set (The raw data for this example is included in the [Resources](#) section):

Table 1. IBM stock price

Date	Open	High	Low	Close
10-Jun-04	90.23	90.75	89.89	90.46
9-Jun-04	89.90	90.55	89.81	90.09
8-Jun-04	88.64	90.50	88.40	90.04
7-Jun-04	88.75	88.99	88.01	88.64
4-Jun-04	87.95	88.49	87.50	87.56
3-Jun-04	87.85	88.10	87.35	87.35
2-Jun-04	88.64	88.64	87.89	87.98
1-Jun-04	88.00	88.48	87.30	88.12

Most data sets have numerical columns, but this one represents a challenge because the x-axis is time data. The following lines tell gnuplot how to read and format the time data on the x-axis (see `help time/data` and `help set timefmt` for more details):

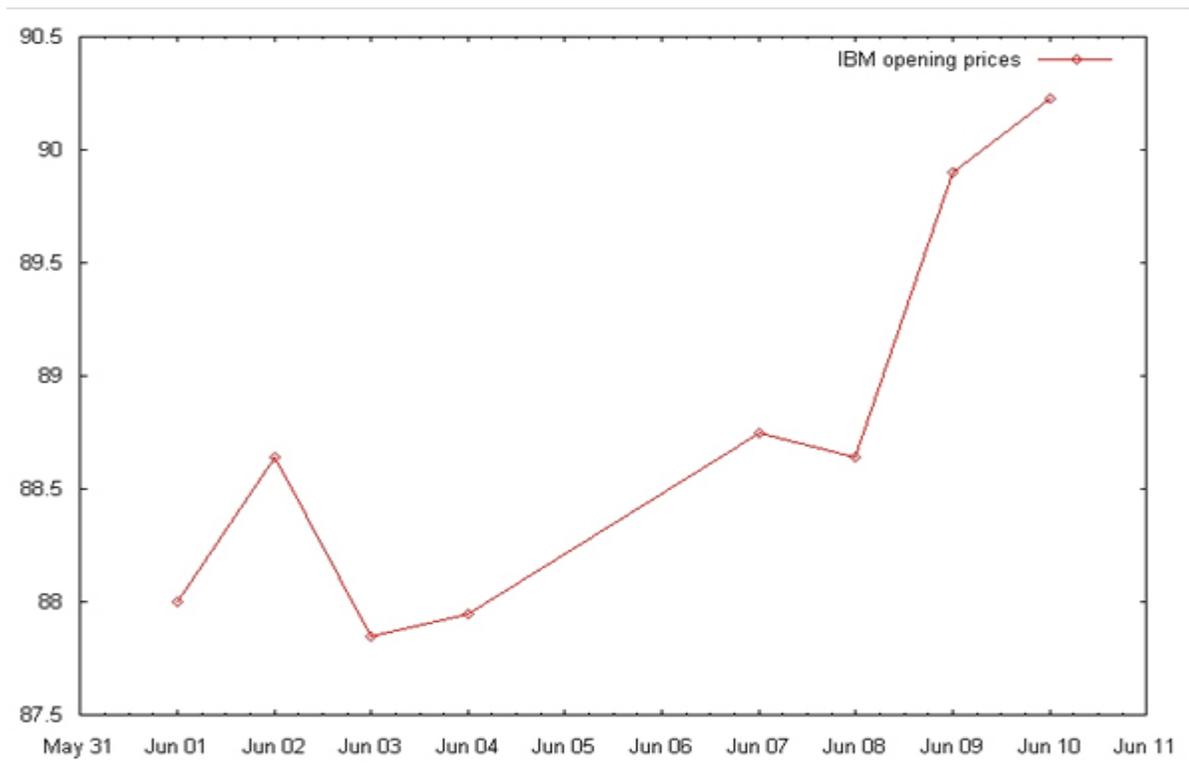
Listing 8. Setting up timeseries data

```
set xdata time          # The x axis data is time
set timefmt "%d-%b-%y" # The dates in the file look like 10-Jun-04
set format x "%b %d"   # On the x-axis, we want tics like Jun 10
```

Once that is set up, we can plot the opening price with the following command. We choose to use straight-line interpolation between the different opening prices, and use the `linespoints` style instead of just points:

```
plot ["31-May-04":"11-Jun-04"] 'ibm.dat' using 1:2 with linespoints
```

Figure 8. Charting IBM stock prices



As always, ["31-May-04":"11-Jun-04"] sets up the xrange explicitly, but is not strictly necessary. Note that the min and max of the range need to be specified as strings that conform to the `timefmt`.

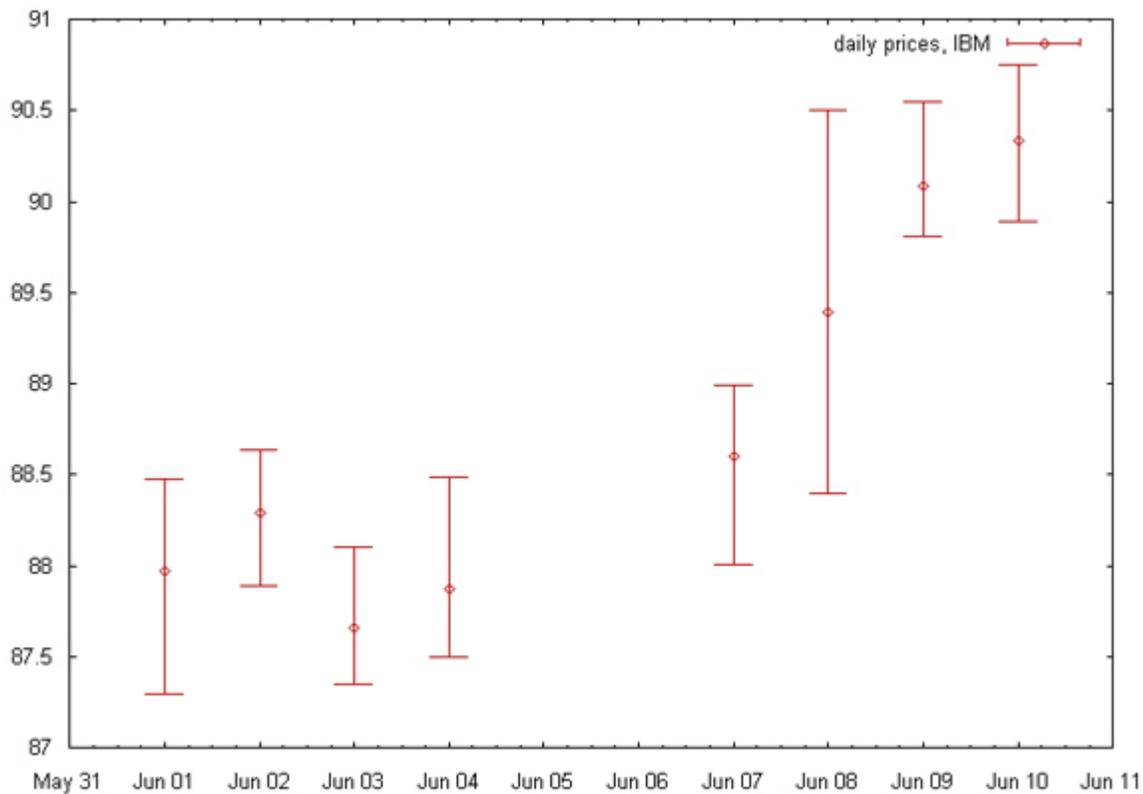
Depending on the plotting style being used, gnuplot may use up to six different numbers in plotting a single point. When a record is read in, the columns are numbered starting from 1. `using 1:2` tells the plot command to use the first column as the x-axis and the second as the y axis. So to plot the closing prices, we would use `using 1:5`.

The column numbers are incremented each time gnuplot sees a new data separator. By default, this is a white space. So if our time format had a space in the middle (for example `10 Jun 04` as opposed to `10-Jun-04`), we would need to use `using 1:4` instead of `1:2`. For the x-axis, we only need to specify the first column where the data starts. Gnuplot is intelligent enough to use the `timefmt` string to read the other columns.

`using` lets you add more than two colons to specify any additional columns that may be needed by a particular plotting style. You can also plot arbitrary expressions using `$n` to refer to the *n*th column. In an entirely cooked-up example, let us suppose the different prices represent different runs of an experiment, and we want to plot the average, the high and low values. The `yerrorbars` style lets you plot the y-high/y-low values, which we specify as `x:y:yhigh:ylow` in the `using` modifier, and calculate the average of the four prices for the y-axis as $(\$2+\$3+\$4+\$5)/4$:

```
plot ["31-May-04":"11-Jun-04"] 'ibm.dat' using 1:($2+$3+$4+$5)/4:4:3 \
title 'daily prices, IBM' with yerrorbars
```

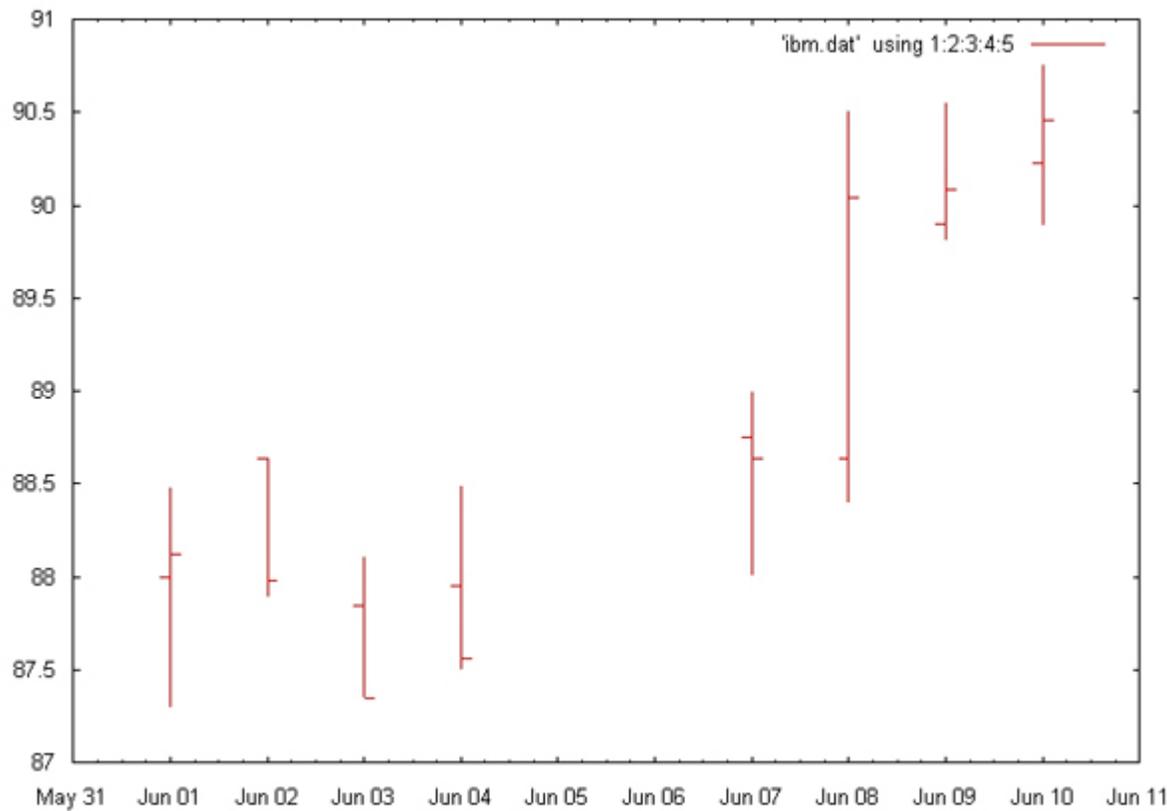
Figure 9. Calculating averages



The errorlines style is similar, but it also draws line segments between the y-points plotted. Gnuplot also has a separate style for plotting finance bars, with the high and low values and tics on the left and right for the opening and closing values, respectively. We use `set bars` to show larger tics for the opening and closing values):

```
set bars 5
plot ["31-May-04":"11-Jun-04"] 'ibm.dat' using 1:2:3:4:5 with financebars
```

Figure 10. Plotting finance bars



You should also look up `help plotting styles` for other relevant styles like candlesticks, boxerrorbars and error plotting styles.

Dealing with real-life data sets

Of course, things can get more complicated in real-life data sets. Having data sets with a non-white space character separating the data points is a very common occurrence. New to gnuplot 4.0, you can specify your own separator using `set datafile separator <string>`. Similarly, you can also specify a comments character other than `#` with `set datafile commentschar`. If you need more control, you can explicitly specify a format string for the `using` modifier. Gnuplot reads in the data using the C standard library's `scanf` function, with the given format string (see `help using` for all the details). I have only mentioned this method for the sake of completeness; most of the data sets that can be read in this way can also be handled using either the `set datafile separator` or by explicitly specifying the column numbers to use in the `using` modifier. For instance, if you have a string in the first column that you need to ignore, plot using `2:3` instead. For complete flexibility, in Unix-like systems with a `popen` function, you can pre-process the file with text processing utilities before it is passed into the plot command. For example: `plot "< awk --f preprocess.awk data.file"`. Unfortunately, this does not work across all systems. A last resort would be to explicitly pre-process the input file before using it in gnuplot.

At times, a real-life data set may not have all the points -- there may be an x-value but the corresponding y-value may be missing for the column you are trying to plot. Gnuplot lets you specify a character string to mean a missing data point. For example:

```
set datafile missing 'NaN' # The IEEE floating point not-a-number
```

Note that, prior to 4.0, the command was `set missing <string>`.

Unfortunately, the way a data point with missing data is handled depends on the `using` specification. See `help using` if you want to know the gory details of the differences between the usages `plot 'file', plot 'file' using 1:2, and plot 'file' using ($1):($2). But to be safe, always specify the columns to use with an explicit using format, and if you want to completely weed out bad data points, do not use the expression evaluation (the third option evaluates the expressions because of the parentheses) -- instead, pre-process the data file and add explicit columns with the result of the calculated expressions.`

At other times, you may end up with some data points that are not valid. You can still plot such data sets if gnuplot can be told to ignore the invalid data points. Gnuplot quietly ignores undefined points, so the trick is to use an undefined y-value like 1/0 when you come across an unsuitable point. The C-like ternary operator comes very handy in expressing this (See `help ternary` if you are not familiar with this function; it is also useful in defining piece-wise functions). Here is an example mentioned in `help using` that lets you plot the second column as a y-value unless the third column is more than 10:

```
plot 'file' using 1:($3>10 ? $2 : 1/0)
```

An advanced example

While this article has exposed you to most of the commands you will end up using commonly, there may be occasions where you need to do more esoteric things. In this section, we briefly discuss some of these concepts, and then include a script from one of the gnuplot demo files (`electron.dem`), which plots an amplitude-frequency response plot of the sort commonly seen in electronics to illustrate their usage.

1. User-defined functions and variables

You can define arbitrarily complex functions of your own such as $A(j\omega)$ below. These can be parameterized (`p1`, `p2`) and you can define the parameter variables later on, before using it in a `plot` command.

2. Complex numbers

Gnuplot understands complex numbers. The notation $\{a,b\}$ represents the complex number $a+bi$. You can use the functions `abs()` and `arg()` to obtain the absolute value and the argument.

3. Dummy variables

When you use `sin(x)` in a plot command, gnuplot uses `x` as the dummy variable and gives it different values to obtain the samples. When writing complex user-defined functions, you can use traditional variable names used in the function definitions and explicitly set the name of the dummy variable to use (for example, `set dummy jw`).

4. The x2 and y2 axes

Apart from the familiar `x` and `y` axis along the bottom and the left edges of the graph, you can also use the top of the graph (`x2`) and the right end of the graph (`y2`) as separate axes. The axes are all independent of each other, and you can set the range, tics, labels, and scaling (for example, `logscale`) separately. Also, the `plot` command can be made to use a different set of axes like `x2y2` by using the `axes` clause. By default, the `plot` command plots against the `x` and `y`-axes. This advanced feature is helpful when you need to simultaneously plot to lines whose ranges are of different orders of magnitude, or have different units. In the example below, the `y`-axis plots the amplitude, which has a maximum of 1, while the `y2`-axis plots the phase, which has a maximum value of 100. If plotted on the same set of axes, the amplitude curve would be barely visible.

5. Logscale

You can plot any of the axes in log scale with the `set logscale` command. This plots the logarithm to base 10 of the values on the axis. You can explicitly set the base to use (for example, `set logscale 2`, or `set logscale xy 2` if setting it only for the major `x-y` axes).

Here is an extract from the demo `electron.dem`, which ships with gnuplot. This script uses all of the advanced concepts discussed above:

Listing 9. Amplitude frequency response

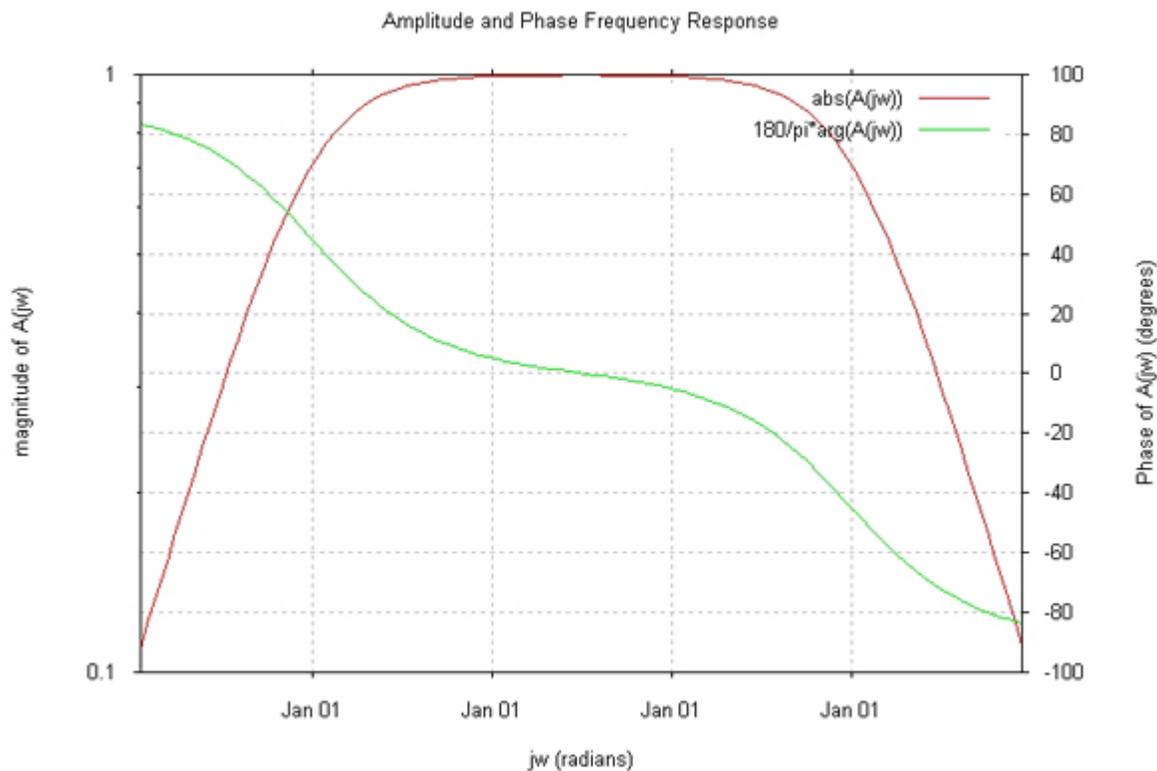
```
A(jw) = ({0, 1}*jw/({0, 1}*jw+p1)) * (1/(1+{0, 1}*jw/p2))
p1 = 10
p2 = 10000
set dummy jw
set grid x y2
set key default
set logscale xy
set log x2
unset log y2
set title "Amplitude and Phase Frequency Response"
```

```

set xlabel "jw (radians)"
set xrange [1.1 : 90000.0]
set x2range [1.1 : 90000.0]
set ylabel "magnitude of A(jw)"
set y2label "Phase of A(jw) (degrees)"
set ytics nomirror
set y2tics
set tics out
set autoscale y
set autoscale y2
plot abs(A(jw)), 180/pi*arg(A(jw)) axes x2y2

```

Figure 11. Amplitude and phase frequency response



Conclusion

In this article, we discussed the intricacies of plotting 2D graphs using the newly-released gnuplot 4.0. Although we touched upon most of the important aspects of using gnuplot, there were a few topics we could not discuss owing to space reasons. Notable omissions range from the very simple plotting of parametric functions (See `help parametric`), and polar coordinates (`help polar`), to curve-fitting, which fits a user-defined curve to a given data set. Curve-fitting is an art, and a whole article could easily be devoted to it, but look up `help fit` and the *beginners_guide and tips* sub-topics to get a start.

Tips for frequent usage

As we have seen, gnuplot is highly customizable. I will conclude with this final tip that shows how to re-use your customizations across several gnuplot sessions: The main command for customization is the `set` command. You can save all sets from the current session using `save set 'filename'.save var <file>` and `save func <file>` save user-defined variables and functions, respectively. But there is no way to save customizations that were passed to a single plot command (for example, the x-range in `plot [-pi:pi] sin(x)`). These files can be read

back using `load <file>`. Gnuplot also looks up a file called `.gnuplot` on startup. It looks first in the current directory, and then in the home directory of the user (The USERPROFILE directory in Windows). If the initialization file is found, gnuplot executes the commands in it. Some users use this for setting the terminal type and defining frequently-used functions or variables.

Download

Name	Size	Download method
source.zip	2KB	HTTP

→ [Information about download methods](#)

Resources

- Visit [the official site for gnuplot](#) and [download the latest version](#).
- Do please note that gnuplot is not GPL'd, nor in any way related to [GNU](#) or the Free Software Foundation. [An older GNU bulletin](#) explains: "Curiously, [gnuplot] was neither written nor named for the GNU Project; the name is a coincidence. Various GNU programs use gnuplot."
- Indeed: gnuplot works exceptionally well in combination with the [GNU plotutils](#); see also the [plotutils documentation](#).
- The official [Gnuplot FAQ](#) is very useful, as is the unofficial [not so Frequently Asked Questions](#) site (the latter has specific information about using gnuplot in scientific papers).
- A [Google search on "gnuplot introduction"](#) or "gnuplot tutorial" yields several short gnuplot 3.7.x tutorials that can be useful, even though they reference the previous, and not the most recent, release version.
- Download the [IBM stock price data \(ibm.dat\)](#) used for the data plots, and the [example TeX file \(input.tex\)](#), which shows how you can include gnuplot plots in a LaTeX report.
- The article "[Data visualization using Perl/Tk](#)" (*developerWorks*, 2003) discusses plotting with Perl.
- For an overview of using ImageMagick from the command line, read "[Graphics from the command line](#)" (*developerWorks*, July 2003).
- "[Introduction to Scalable Vector Graphics](#)" (*developerWorks*, 2004) shows how to to easily generate graphics (such as graphs and charts) from database information, and how to to add animation and interactivity to graphics.
- Find more resources for Linux developers in the [developerWorks Linux zone](#).
- [Browse for books](#) on these and other technical topics.
- Develop and test your Linux applications using the latest IBM tools and middleware with a [developerWorks Subscription](#): you get IBM software from WebSphere, DB2, Lotus, Rational, and Tivoli, and a license to use the software for 12 months, all for less money than you might think.
- Download no-charge trial versions of selected developerWorks Subscription products that run on Linux,

including WebSphere Studio Site Developer, WebSphere SDK for Web services, WebSphere Application Server, DB2 Universal Database Personal Developers Edition, Tivoli Access Manager, and Lotus Domino Server, from the [Speed-start your Linux app](#) section of developerWorks. For an even speedier start, help yourself to a product-by-product collection of how-to articles and tech support.

About the author

Nishanth R. Sastry received his bachelor's degree in Computer Science and Engineering in 1999, graduating with distinction from the R.V. College of Engineering, Bangalore University, India. In 2001, he obtained a master's degree in Computer Science from the University of Texas at Austin. He is an alumnus of IBM's Extreme Blue internship program, and is now a Staff Software Engineer working on IBM Lotus Workplace. You can reach him at nishanth_sastry@us.ibm.com.