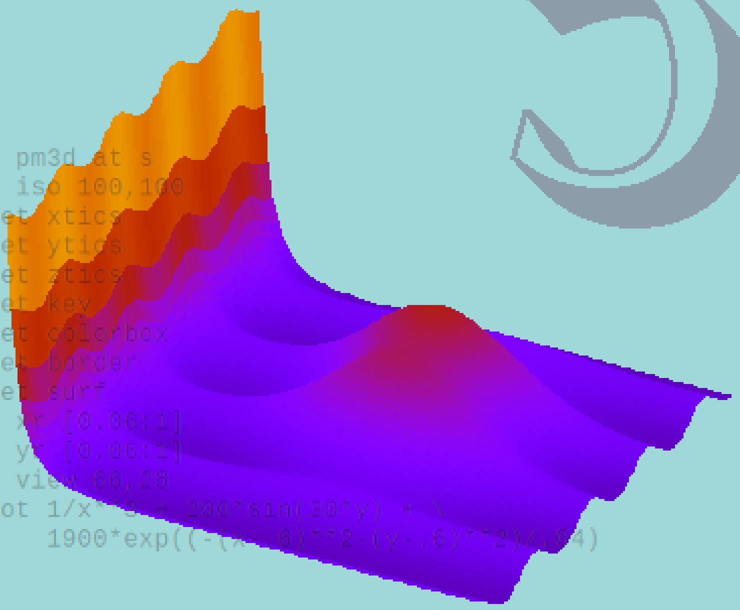
gnuplot 5



```
set pm3d at s
set iso 100,100
unset xtics
unset ytics
unset ztics
unset key
unset colorbox
unset border
unset surf
set xr [0.06:1]
set yr [0.06:1]
set view 66,28
splot 1/x**3 + 200*sin(30*y) + \
      1900*exp((-(x-.6)**2-(y-.6)**2)/.04)
```

Lee Phillips

A visual guide to the most powerful version yet of the go-to plotting system for scientists, engineers, and analysts.

gnuplot 5

by Lee Phillips

Alogus Publishing (a division of the Alogus Research Corporation)

First Edition, 2017

Published by Alogus Publishing, a division of the Alogus Research Corporation.

https://alogus.com/publishing/

Contents

Preface

This book is designed to show you how to make the graph or visualization you want as quickly and painlessly as possible. It is organized to lead you directly to the gnuplot script that will create the result you have in mind; these are working scripts ready for you to copy, paste, and modify for your problem. With this in mind, I have arranged the contents, as far as possible, into an interactive reference that will allow you to touch the result you are aiming for and be taken directly to a working script that produces that result. After years of using and helping others to use gnuplot, I've learned that starting with a script that gets you 90% of the way there, and making obvious changes, is far easier than trying to remember all of the details of gnuplot's syntax and the tricks, accumulated over the decades, to bend it to your will. This is especially true if you turn to gnuplot occasionally, instead of working with it every day.

That being said, this is not merely a collection of recipes. Through the use of progressively more complex examples, and clear explanations of how things work, this book will provide you a thorough understanding of the program. The sections on using gnuplot with other programming languages, integrating it into LaTeX in various ways, and using it on the web, are places where we will need to go a bit beyond the simple example → script structure, but full, working examples will still be provided.

Gnuplot has a built-in help system and official documentation; there exists an excellent book about data analysis, with a focus on gnuplot, and a paperback reference manual. There are many websites, going back decades (including mine), that offer tips for doing all kinds of things with gnuplot. All of these resources are gathered together in this book, in a section at the end as a convenience for the reader. Despite the abundance of material available about gnuplot, however, the book you are reading now serves a unique purpose. This book is mainly intended for the user who needs to create or modify a graphic for a presentation or a paper, and, as is usually the case, is in a bit of a hurry. There us no time to read a thick reference manual while trying to guess which section actually tells you how to do what you need to do. You might try a search on the web, but you may not even know the name for the type of graph or effect you have in mind, and, if you do find something that looks useful, it's likely to be far out of date. Gnuplot 5 is a big advance over even the most recent major version, and has been available for only a small fraction of gnuplot's lifespan to date; consequently, the vast majority of online gnuplot tips refer to older versions.

I hope you will find this book to be a valuable resource of a kind that does not exist anywhere else. All the features of gnuplot 5 are gathered together here in one unified, organized document. One of the unique features is a visual index, where you can browse for the type of visualization you need and click on it to be taken to a working, tested gnuplot script. You don't need to guess what jargon to search for, or even know ahead of time that gnuplot can create the graph that you have in mind.

My publisher and I have decided to issue *gnuplot 5* exclusively as an electronic book. This decision was partly due to my experience with my previous *gnuplot Cookbook*: the electronic versions were more useful than the paper version, in which the various types of figures did not uniformly reproduce well on paper. Another consideration is that this is not a book that one is likely to spend much time with on the beach or while relaxing in a hammock, but rather to use while one is working at the computer, actually engaged in the process of making graphs; therefore having the book in electronic form, with the ability to directly copy and paste solutions, is bound to be more useful. Extensive use is made of hyperlinks to make navigation as easy and useful as possible. Colored text indicates a hyperlink, with different colors used to distinguish between internal and external links, similarly to the styling of most web pages.

We are aware of the main drawback of the PDF format: the text does not flow to fit the viewing window; the page's aspect ratio is fixed. Unfortunately, the usual attempt at a remedy is worse than the disease: the typography of non-pdf ebooks ranges from mediocre to unreadable. This may not be a fatal flaw for books of pure prose, and I myself have read entire novels in conventional ebook form on phones, with pleasure. But this type of book, with a mix of code samples, prose, equations, and various types of color figures that must be kept in the correct association with the text, requires the exacting typographical control that is only possible using PDF. We have tried to compensate for the rigid nature of the format by offering several versions of the book customized for different screen dimensions. Your purchase entitles you to try out and download any and all of these, so that, whether you are using a laptop or a tablet, you should be able to find a version that works well on your device.

We use no DRM whatsoever. Your purchase gives you the right to use the book online as often as you please, and to download it and make as many backup copies, on servers (the "cloud") or on your own media, as you desire. You may install and use the book on any and all of your devices. We only ask that you remember that this is a copyrighted work, created by expending a great deal of time, effort, and money, and refrain from any type of public distribution. Not only would this be illegal, but would make it harder for us to support the book in the future, to keep it available online, to incorporate corrections in later editions, etc. That being said, neither the publisher nor I have any problem with you lending a copy of the book to a colleague; it does belong to you, after all. If you are reading a copy that you've borrowed, and think that you would like to hold on to it for a while, please consider visiting the publisher's website and purchasing your own copy. We've tried to keep the price low enough to keep it within reach even of graduate students. Not only is buying your own copy the right thing to do, but it gives you the opportunity to give us your email address so we can keep you informed (if you want) of updates, corrections, and online-only extra material as we develop it.

About the name

The name of the program starts with a lower-case letter: "gnuplot"; therefore so does the title of this book. To avoid a certain awkwardness, we've decided to capitalize the name of the program at the beginning of sentences.

The name does not refer to the GNU free software project.

Why gnuplot?

Gnuplot is a free, open-source plotting program that has been widely used since 1986. It forms the graphics back-end by many other programs, so you may have used gnuplot without knowing it.

Gnuplot was originally intended to visualize scientific data, but its use has expanded to encompass every domain where a sophisticated, accurate, and efficient plotting are required. Gnuplot is used in the sciences, engineering, mapping, business, finance, and computer server and network performance monitoring.

Gnuplot excels at complex three-dimensional graphing and at the rendering of surfaces and contours. It can produce almost any type of graph imaginable (even pie-charts, with coercion, if you insist) for a dizzying array of output devices. You can save your plots in any type of file format that you can imagine. It can be installed on almost any type of computer system in current use; there are binaries available for Windows and the sources can be compiled on most reasonably modern machines. Gnuplot can easily be automated. It has its own scripting language that can be used for single plots, for analyzing data, and for sophisticated programming to handle everything from data streams to animation. Gnuplot can also be controlled from any other programming language, using either a custom library or communication over a socket interface.

Gnuplot can also be folded into various publishing and documentation workflows to help create professional books, papers, and online documents.

No specialized knowledge is required to make use of this book; although we may take examples from various fields, they are incidental to the examples, which are focused on creating particular types of graphs. The examples of controlling gnuplot from programming languages teach general approaches that can be understood even if you don't have experience with the languages used.

Lee Phillips

Chapter 0

Installation

This is a short chapter explaining how to install gnuplot on the most common operating systems. If you already have gnuplot version 5 installed and working, you might as well skip this chapter. You may already have gnuplot 5 installed, if you have a recent version of a program that uses it as a component, for example Maxima or Octave. If this is the case, using that version may be easier than installing from scratch.

Linux

The package management systems of many Linux distributions include the gnuplot binary, but the version is usually somewhat old. In particular, version 5, the subject of this book, as not yet made it into the latest edition of some popular distributions. Although you can certainly make good use of a version from the 4.x series, and while much of the content of this book will still apply, it's worth it to acquire the latest version. There are many useful features in gnuplot 5 that make it easier to use and more powerful. Also, some of the syntax has changed, so some of the scripts in this book will not work with earlier versions.

To get the lastest version of gnuplot running on your system, download the source code from the official repository, and configure and compile by following the inluded instructions. These are universal sources that should work on any supported operating system. You may have to attempt the compilation several times as you discover you are missing some dependencies (the Pango and Cairo development libraries, for example), but these can all be added through your package manager. After compilation has completed, check the logs or give the command `set term` to gnuplot to make sure that your desired output devices are included. For high quality output for print or electronic publication, you should see one or both of the pngcairo or pdfcairo terminals in the list. Finally, check that you can use the up-arrow and other readline affordances at the gnuplot interactive prompt; if this doesn't work, then a compatible readline library was not found on your system during compilation. Check the compilation log for more details.

OS X

Compilation on the Macintosh is typically more problematic than on Linux, due to the presence of old or incompatible libraries on the system. Apple, for a while, even included a mislabeled, nonstandard readline library. It is, however, possible. An easier route is to install via Homebrew, the "missing package manager" for the Macintosh. Perhaps even easier is to download Octave and look around in the .dmg file for the included gnuplot binary. At the moment, however, there is an immediate solution: at the gnuplot homepage there is a link to "contributed executables for OSX" that includes .dmg archives for the most recent versions of gnuplot.

Windows

The Windows user can try to compile from sources (see the **Linux** section above), but there is a simpler way to install: the download page usually includes a link to a compilation of a recent version. The .exe can sometimes be found within the “testing” folder.

Chapter 1

2D Plots

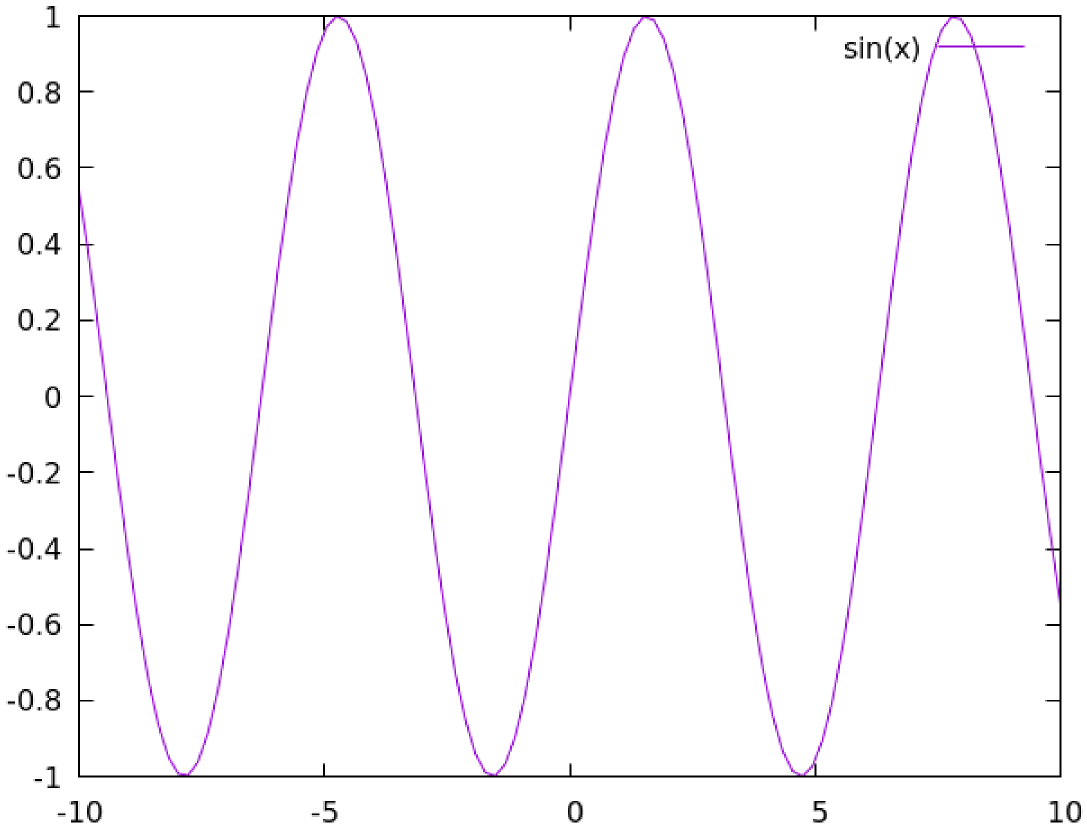
This chapter covers various types and styles of two-dimensional (2D) plots in gnuplot. A 2D plot is a visual description of the relationship between two variables, whether that relationship is described by a mathematical function or a set of data. As in future chapters, we'll start with simple examples and gradually get more complex, showing, along the way, how to customize the graphs' appearance and the information that appears on it. As new commands or syntax elements are introduced, we'll highlight those elements in the code listings to make it easy for the eye to pick out the new material. We'll do this in a way that doesn't affect your ability to copy-and-paste the code to get a working gnuplot script, ready to be adapted to your application.

You can try out these scripts in two ways: you can save each script to a file, and tell gnuplot to execute it, or you can paste the script into the interactive prompt. The latter way may be better for experimenting and learning the system, but each way comes with its own caveat. If you are saving the scripts to files, you can run them by typing `gnuplot file`, substituting the name of the file for `file`. If you do this, however, you may not see any output, or get a brief glimpse of your plot before it vanishes. This is because the plot window for most on-screen "terminals", as gnuplot calls its output devices, goes away when the gnuplot process exits. To prevent this, invoke the program with `gnuplot -persist file`. The caveat that applies to using the interactive prompt is this: some of the lines of code in our examples change various settings. These changes will persist in your session, but each script is intended to be self-contained. Therefore it's probably best to give the command `reset` before starting a new example; this will wipe out any settings that you've altered, and is equivalent to ending the session (with `ctrl-D` on Linux and other Unix-like systems) and starting another one.

Plotting a Function

Here's a mathematical function that you've probably seen before. Gnuplot knows quite a lot of math, so we can just say its name to get a graph:

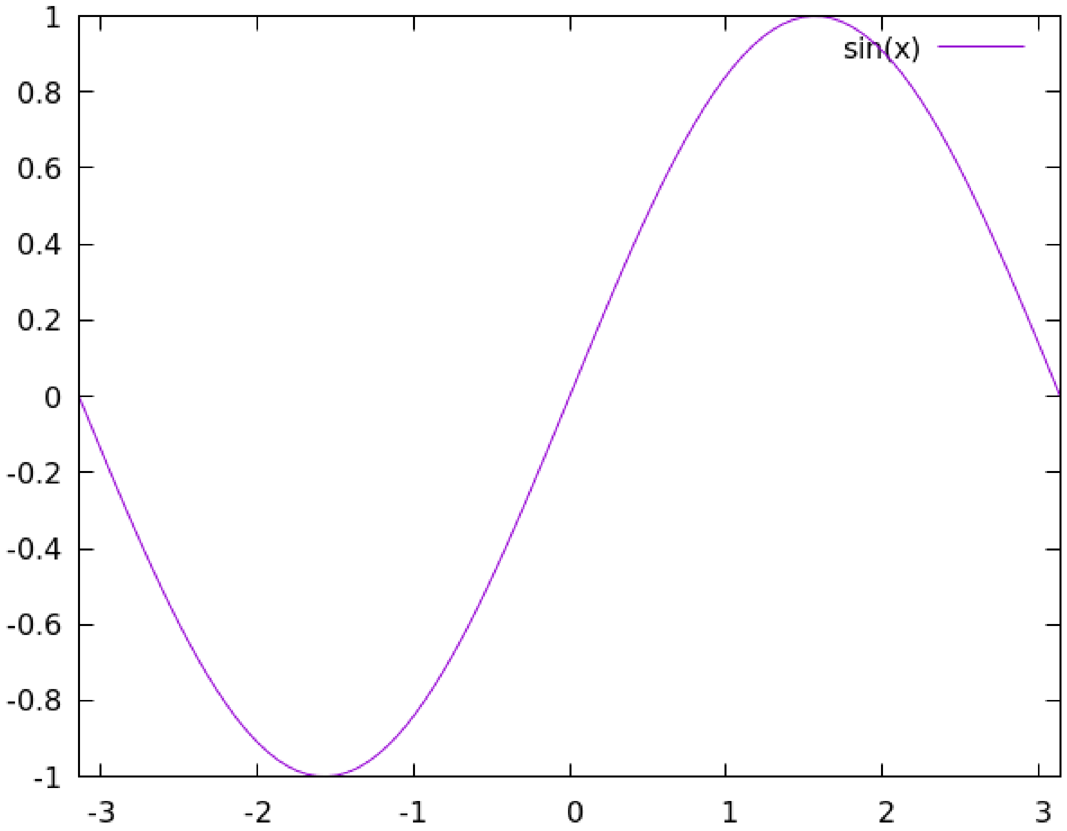
```
plot sin(x)
```



Setting Ranges

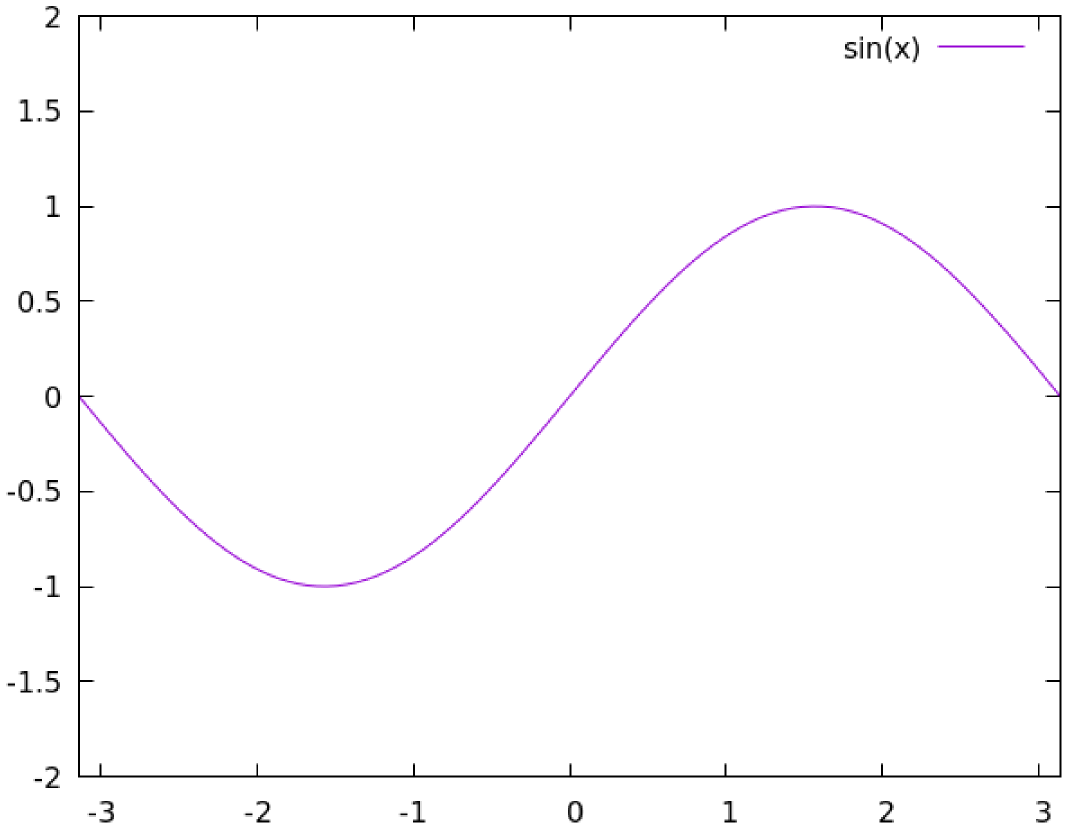
That was simple. Notice how gnuplot decided to plot our function from -10 to +10. That's the default, which we got because we didn't ask for any particular range. Gnuplot also set the y-axis limits (the range of the vertical axis) to encompass the range of the function over that default x-axis domain. Let's take control of the limits on the horizontal axis (the new command is highlighted). Gnuplot happens to know what π is (but doesn't know any other transcendental numbers).

```
set xrange [-pi : pi]
plot sin(x)
```



You may want to set the range of the vertical axis as well, either to show only a part of the function or data, or to leave extra room on the graph.

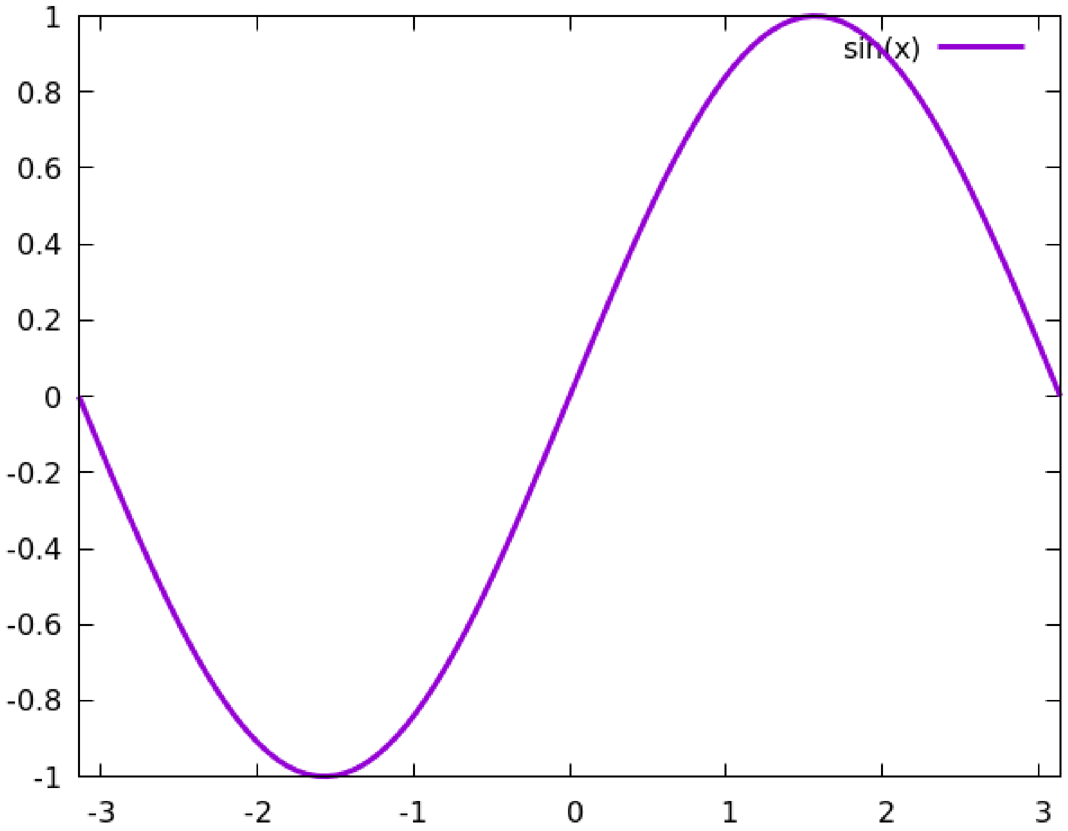
```
set xrange [-pi : pi]
set yrange [-2 : 2]
plot sin(x)
```



Changing the Linewidth

The default styles chosen by gnuplot are not ideal. Fortunately, we can change everything, and make gnuplot's output look any way we want. We'll defer changing the styles of labels, including tic labels, to a later chapter. But for now, let's make the plot curves a bit thicker. The `set lw` command set the **linewidth**; the parameter is the multiple of the default width for the terminal in use.

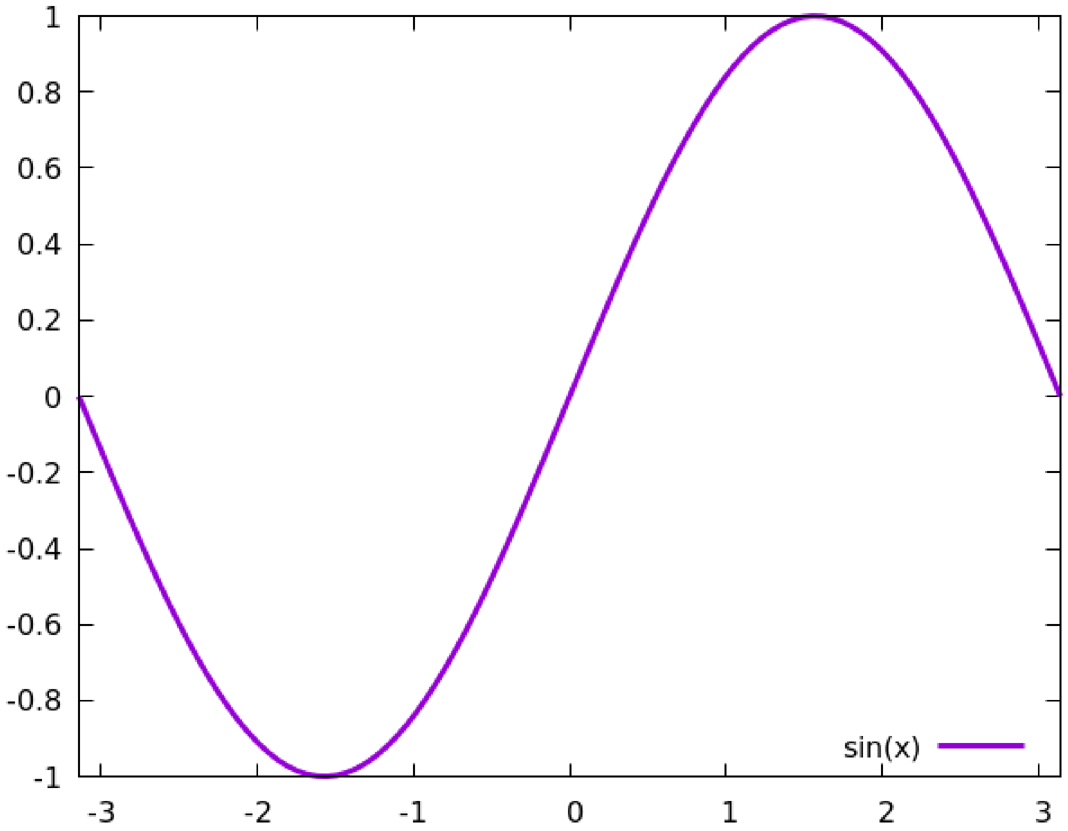
```
set xrange [-pi : pi]
plot sin(x) lw 3
```



Positioning the Key

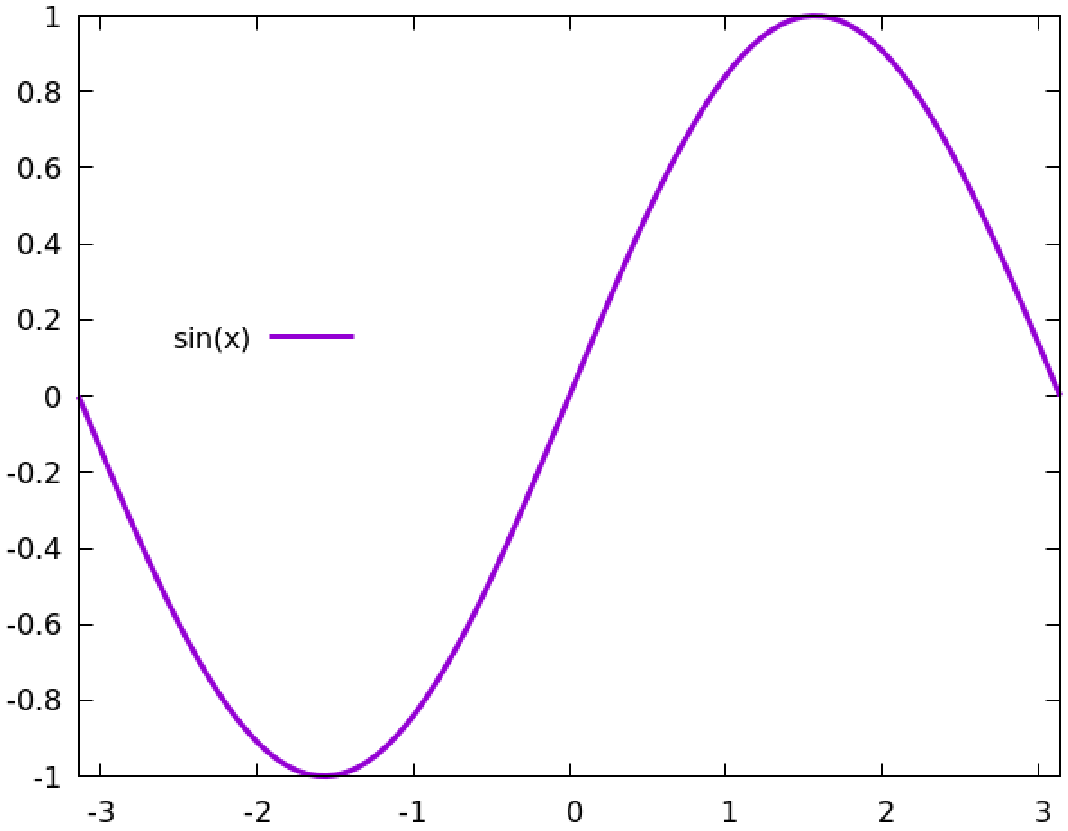
Notice how the automatically generated legend, or what gnuplot calls the “key”, also displays the same styling details we give to the curves. And, speaking of the key, now that we’ve reverted to using the entire plot height, notice how the curve collides with it. Gnuplot positions some things automatically, but the key is not one of them. Gnuplot provides several ways to position the key. Here’s the simplest:

```
set xrange [-pi : pi]
set key bottom right
plot sin(x) lw 3
```



You can probably guess what you get with various combinations of `bottom`, `left`, `right`, and `top`. If you need finer control over positioning, you can refer to one of gnuplot's coordinate systems (as in all the code examples, the new commands are highlighted):

```
set xrange [-pi : pi]
set key at graph 0.3, 0.6
plot sin(x) lw 3
```

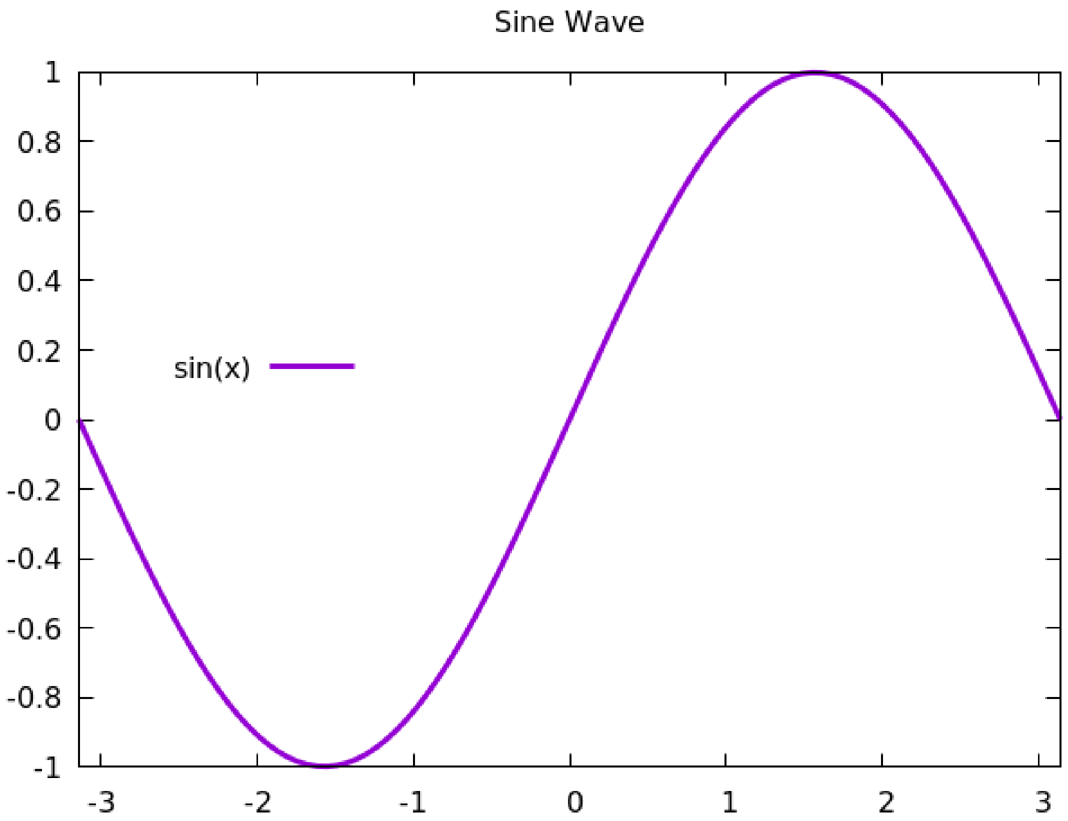


The phrase `at graph 0.3, 0.6` positions the key at location x = 0.3 and y = 0.6 in graph coordinates, which is a coordinate system where (0,0) is at the bottom left of the actual graph (not the screen on which the graph is drawn). Say `help coordinate` to invoke gnuplot's help system and get a rundown of the *five* coordinate systems available to you.

Defining a Graph Title

In a later chapter we'll find out how to style the text used in the key and elsewhere. For now, here is how to add a title to the graph:

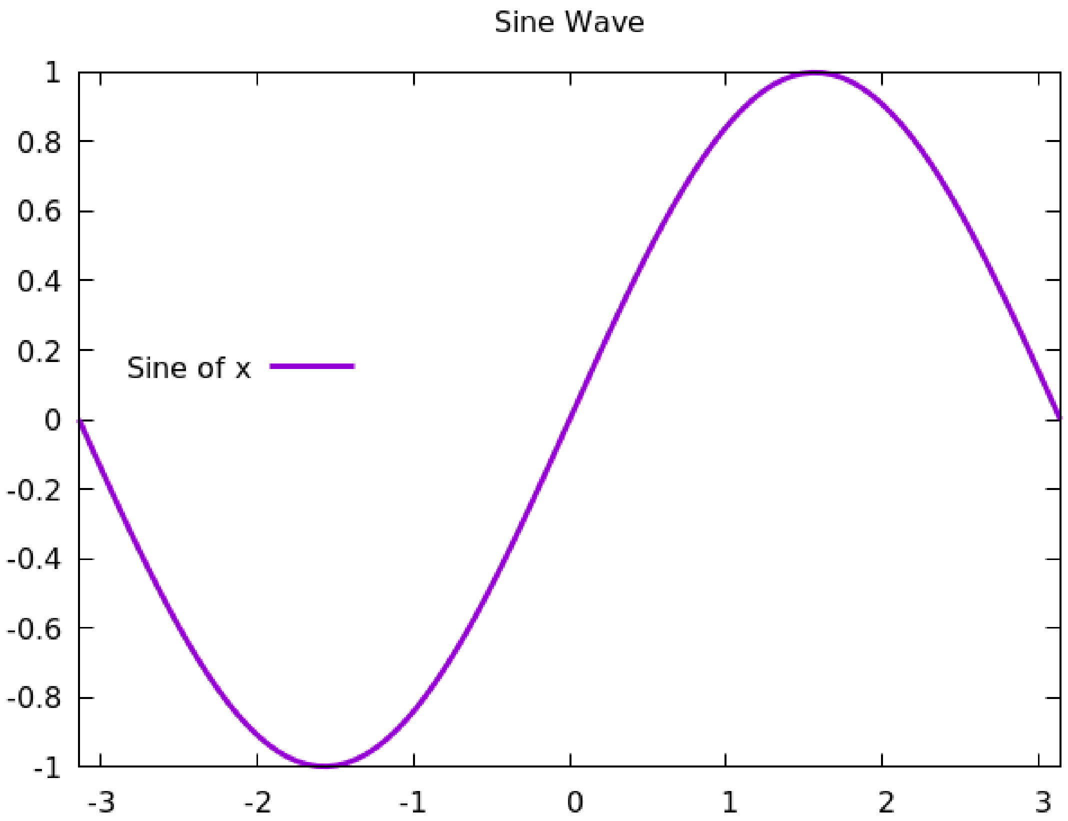
```
set xrange [-pi : pi]
set key at graph 0.3, 0.6
set title "Sine Wave"
plot sin(x) lw 3
```



Titling Individual Curves

You can use Unicode characters in your titles, and anywhere you specify text. We often want to supply individual curves with titles as well:

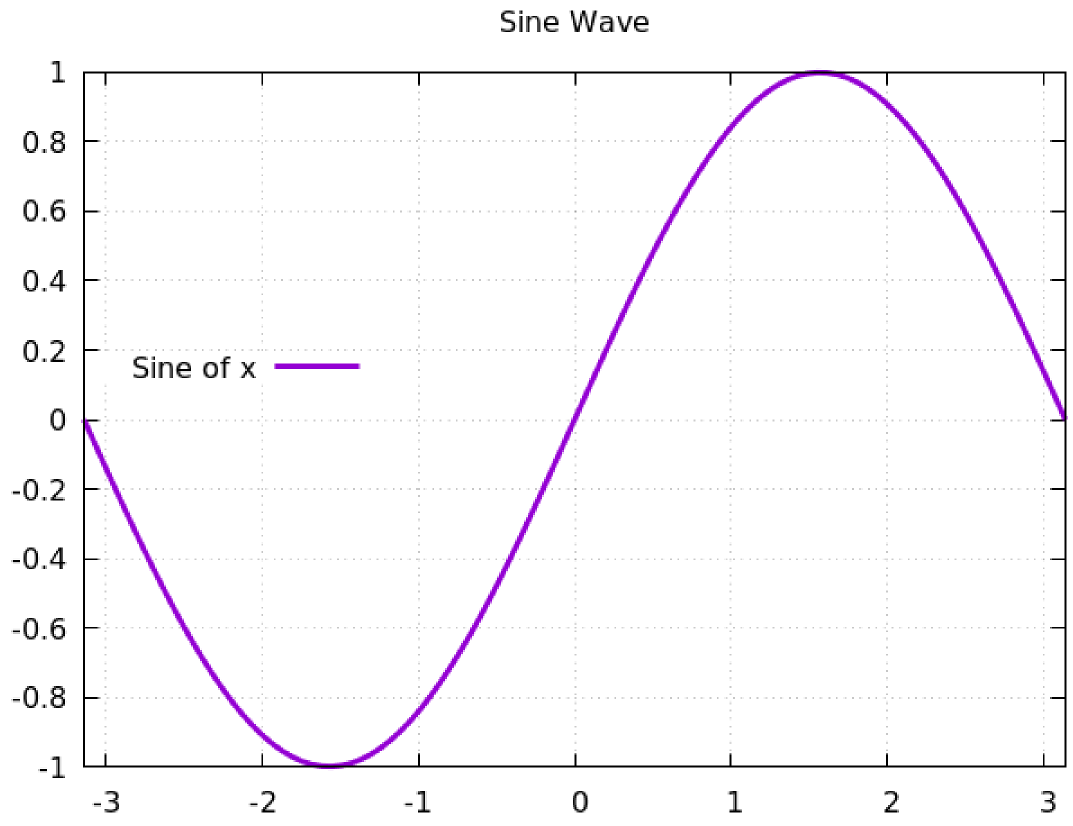
```
set xrange [-pi : pi]
set key at graph 0.3, 0.6
set title "Sine Wave"
plot sin(x) lw 3 title "Sine of x"
```



Grid Lines

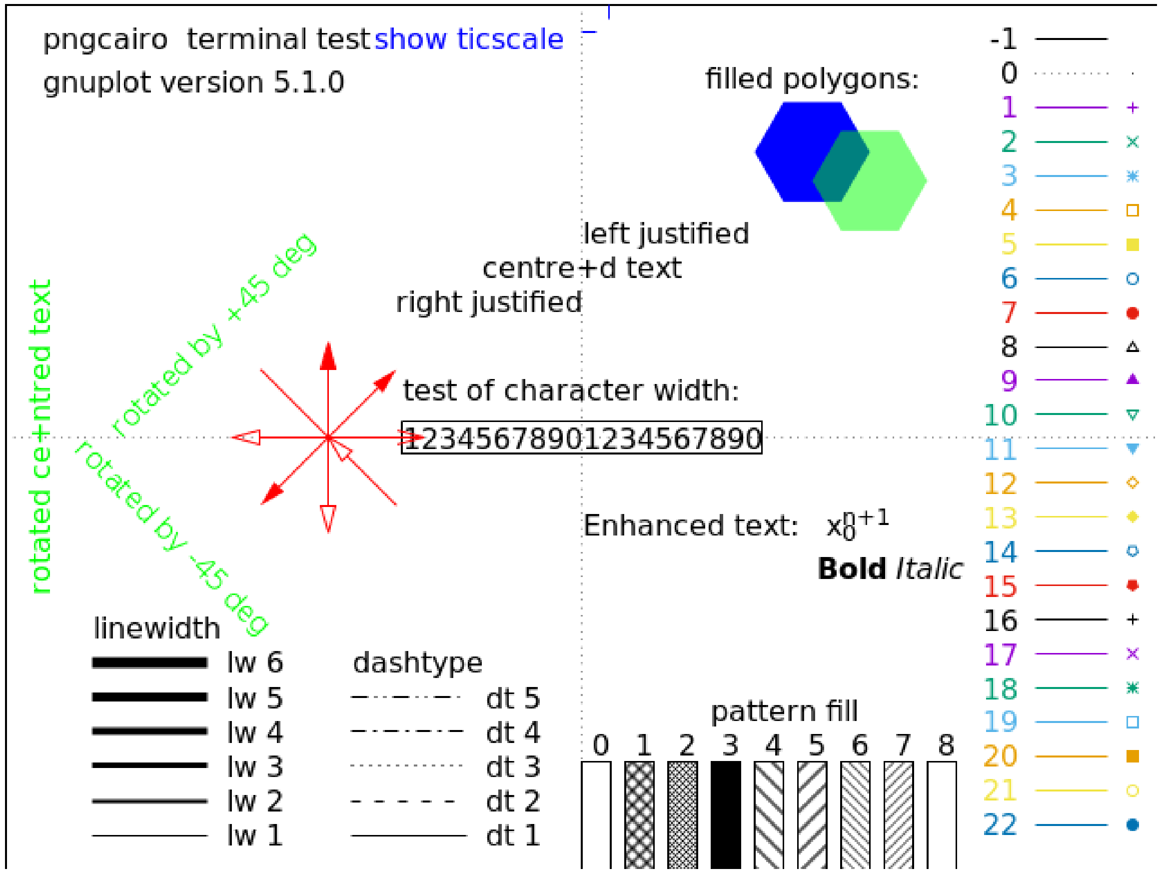
To apply a grid to your graph, use the new command given below. The gridlines will, by default, extend the major axes tics; in a later chapter we'll learn how to get total control over our tics.

```
set xrange [-pi : pi]
set key at graph 0.3, 0.6
set title "Sine Wave"
set grid
plot sin(x) lw 3 title "Sine of x"
```



The gridlines can be styled just as the plotted curves can. In fact, depending on your terminal, the gridlines may be so faint as to be almost impossible to see until you apply some styling to make them more visible. To make it easier to style lines and curves, it's time to introduce the concept of the "linetype" in gnuplot. Each terminal has a set of predefined linetypes (abbreviated `lt`). To see what they are, and to test some of the other capabilities of the terminal you're using, just enter this command:

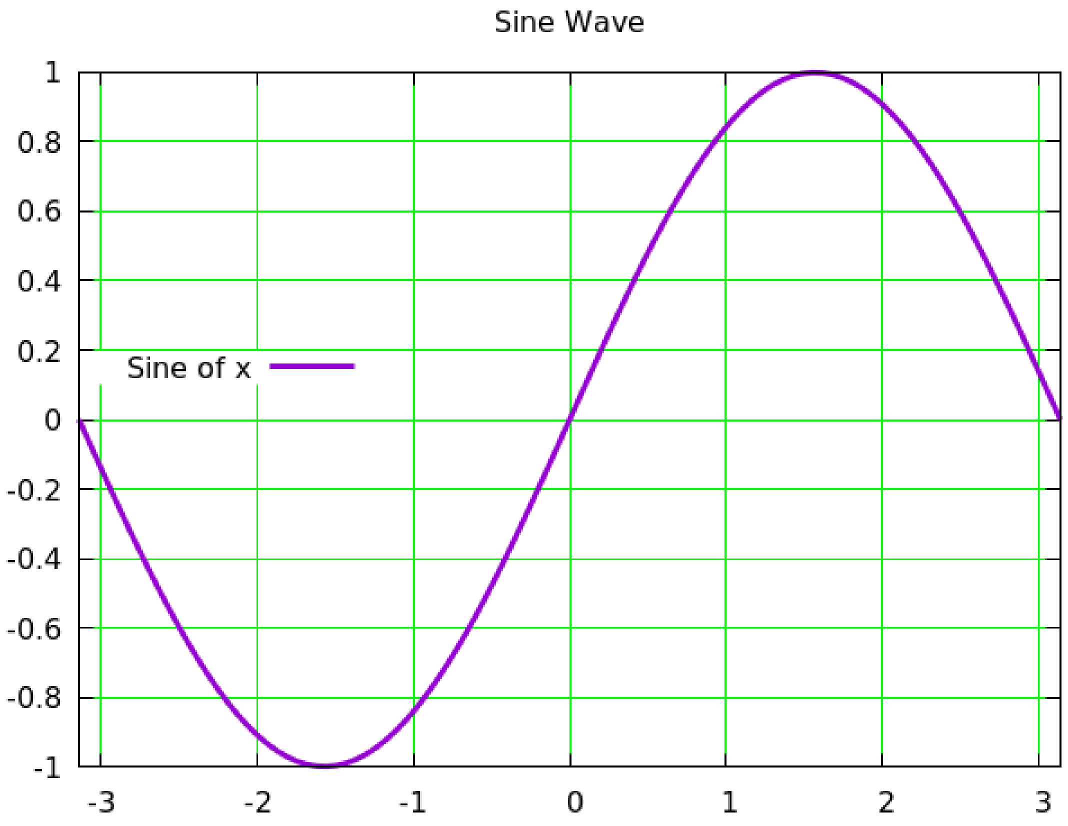
```
test
```



Linetypes

Let's say that you want solid gridlines. Referring to the test plot we just made, you can see that the solid line is linetype 1 (it may be something different on your terminal). Here is how we get a grid with solid, green gridlines:

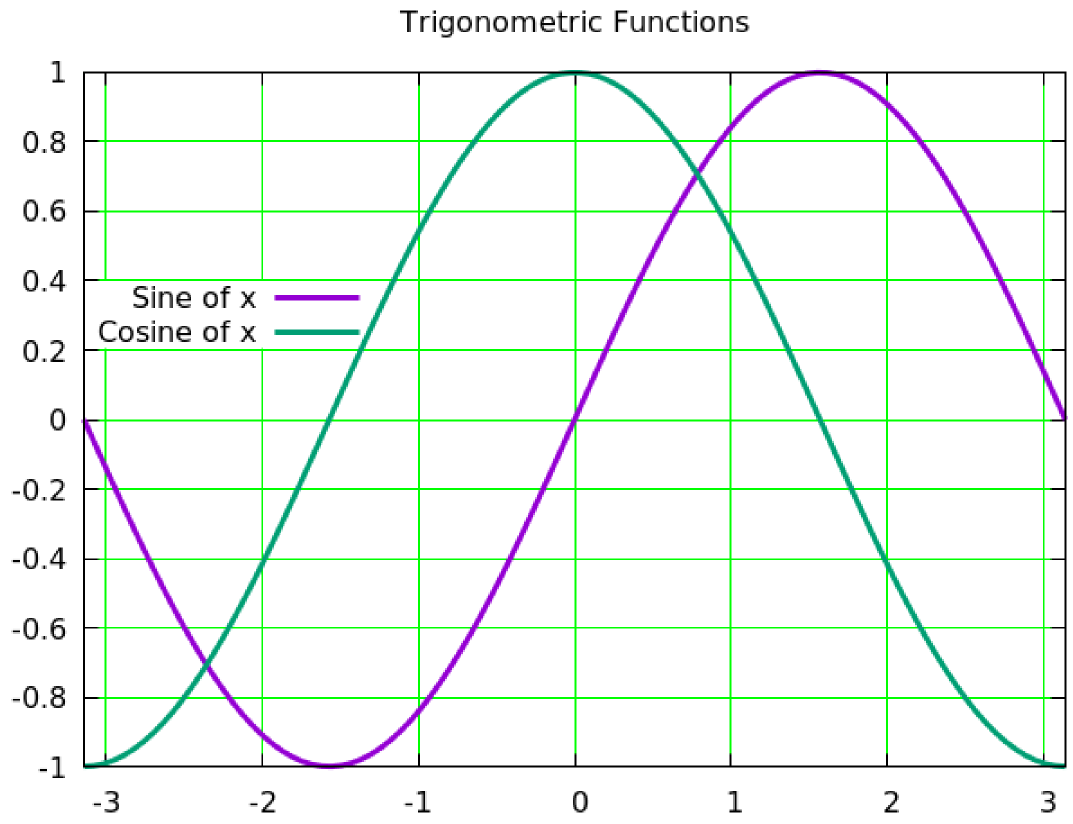
```
set xrange [-pi : pi]
set key at graph 0.3, 0.6
set title "Sine Wave"
set grid lt 1 lc rgb "green"
plot sin(x) lw 3 title "Sine of x"
```



Plotting Multiple Curves

To see the list of color names that gnuplot accepts, just type `show colors` at the prompt. We'll defer a complete discussion of setting styles and colors to a later chapter. To plot more than one curve on a graph, just separate the functions (or data files, which we'll see later in this chapter) by commas, and gnuplot will plot them in a sequence of colors, putting them on the key so you can identify them.

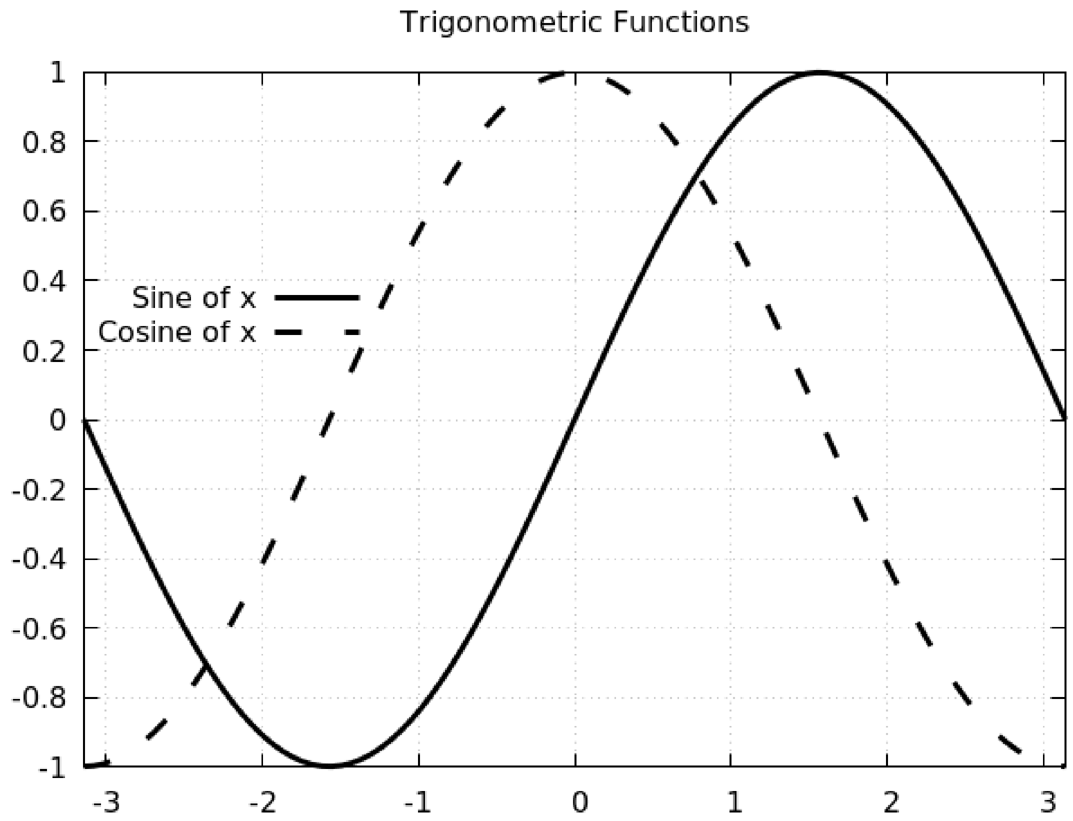
```
set xrange [-pi : pi]
set key at graph 0.3, 0.7
set title "Trigonometric Functions"
set grid lt 1 lc rgb "green"
plot sin(x) lw 3 title "Sine of x", cos(x) lw 3\
   title "Cosine of x"@
```



Monochrome

When preparing a paper for publication, you may have to create black and white renditions of your graphs for the print version. The script here shows the command you need; as you can see, gnuplot substitutes a sequence of dash styles in place of a sequence of colors.

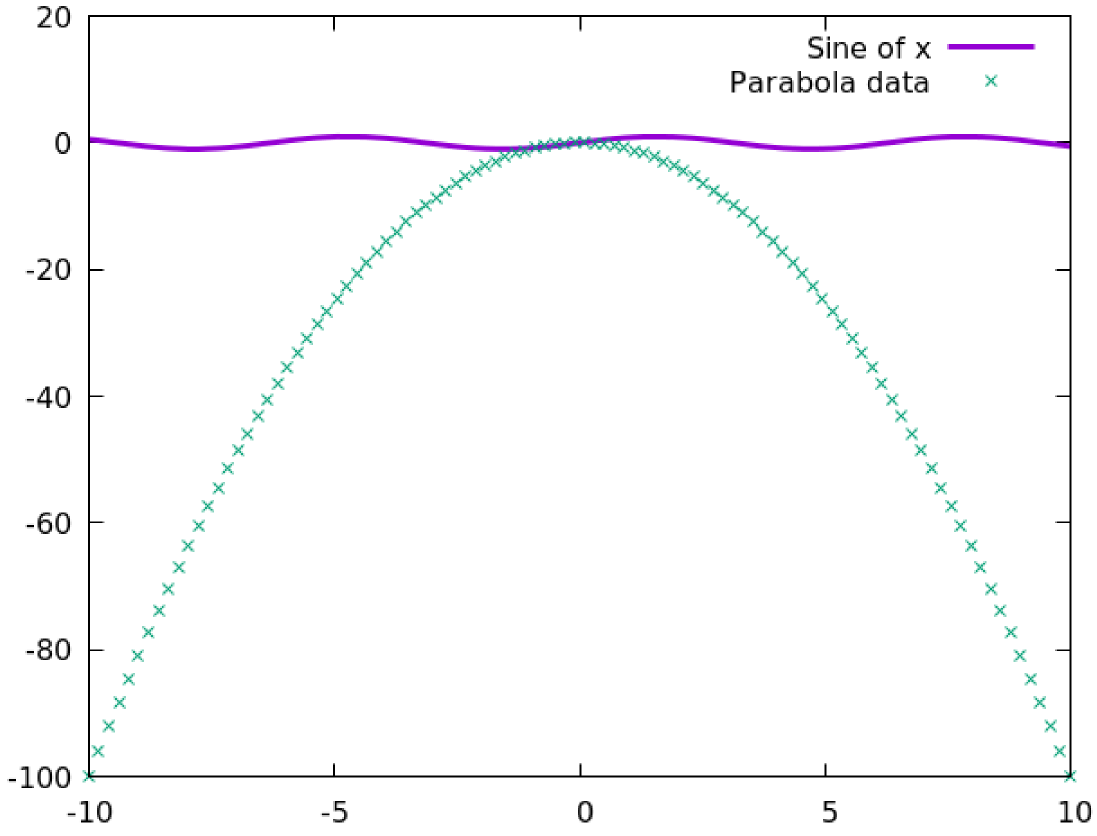
```
set monochrome
set xrange [-pi : pi]
set key at graph 0.3, 0.7
set title "Trigonometric Functions"
set grid
plot sin(x) lw 3 title "Sine of x", cos(x) lw 3\
   title "Cosine of x"
```



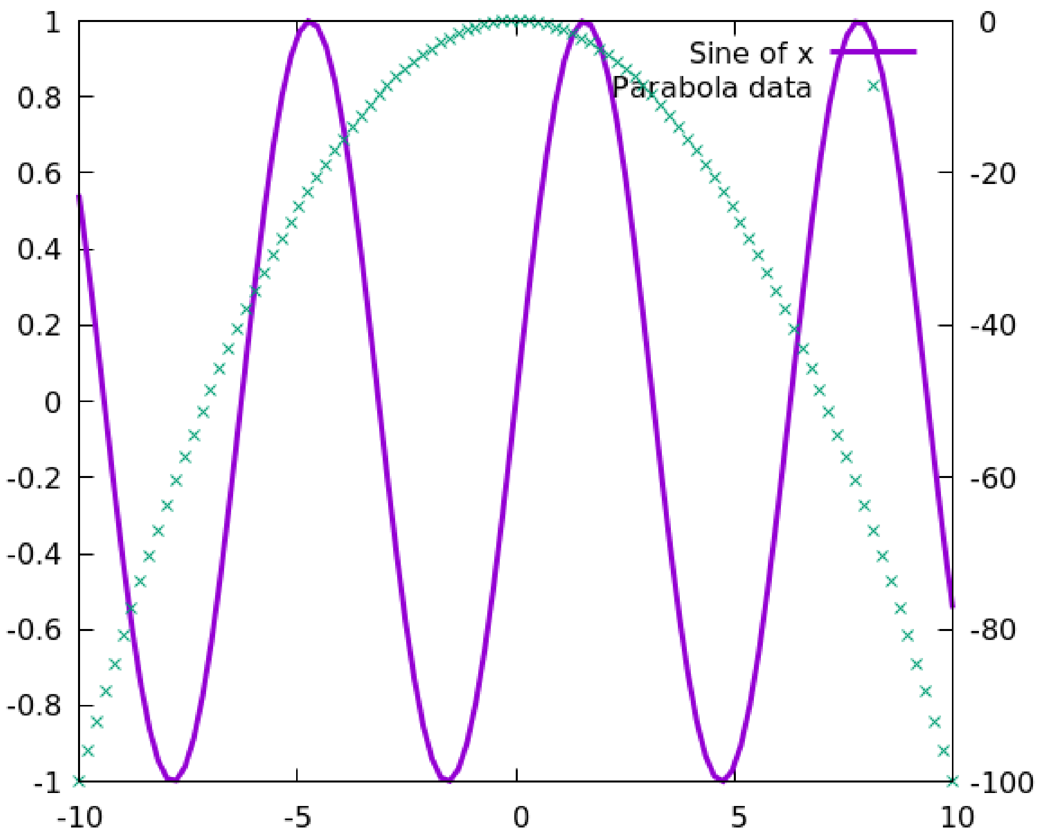
Creating and Plotting Data Files

Of course, gnuplot can plot data from files (and other sources, too, as we'll see in subsequent chapters) as well as mathematical functions. In order to experiment with this, we'll need at least one data file. You could make it by hand with a text editor or write a program in your favorite language to generate some data, or use some that you have available. But gnuplot can make data files itself. To make a file with data that forms a parabola, execute the following script. We've included comments (they begin with the "#" character; everything after this character is ignored by gnuplot) to explain what the new commands do.

```
set table "parabola.dat" # Save numbers to a file.
plot -x**2
unset table # Go back to normal plotting.
plot sin(x) lw 3 title "Sine of x", \
   "parabola.dat" title "Parabola data"
```



After executing the previous script you should have a file called “parabola.dat” in the directory in which you started gnuplot. Keep it around so we can use it with later examples. It will serve, also, as a convenient reference for the format that gnuplot expects when plotting data files. In the last line, after our familiar sine curve, we’ve plotted the numbers read from the file given inside quotation marks. This final line also shows how we can break long lines in scripts into two by using a backslash. Gnuplot has chosen markers, rather than a continuous line, to help indicate that we’re looking at data, but we can change this.



Using a Second y-axis

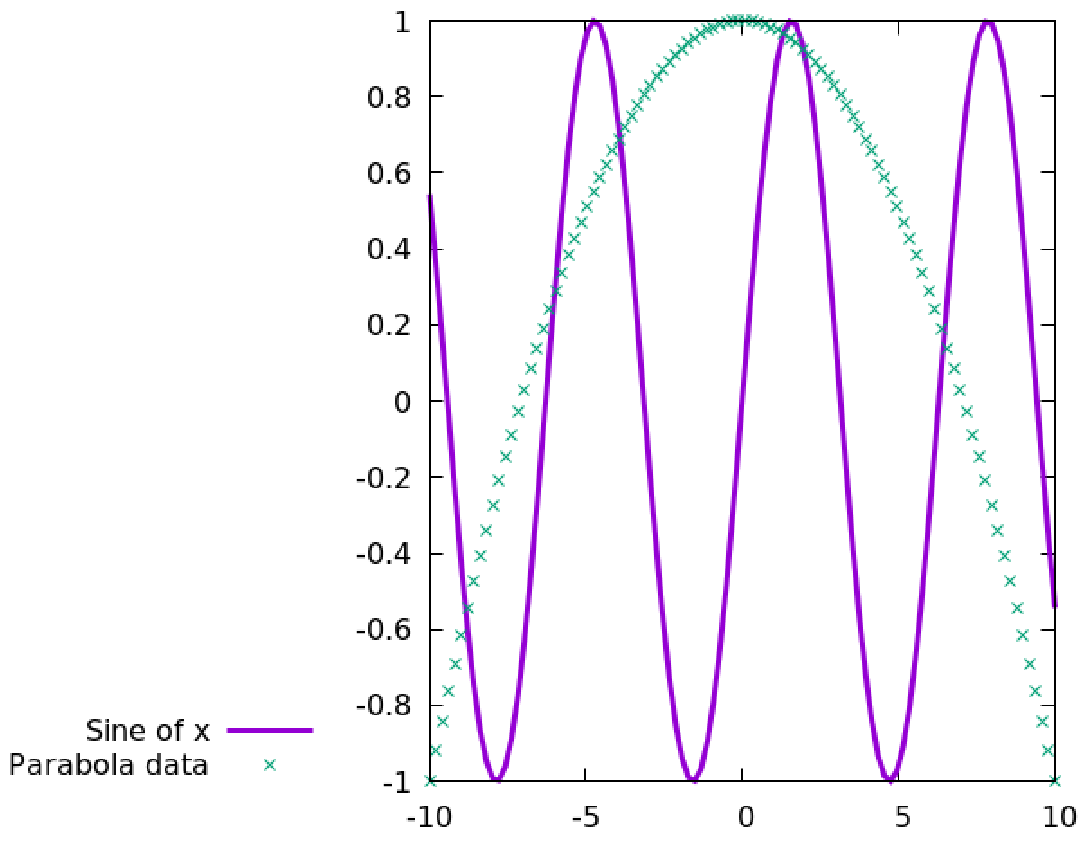
First, however, notice how the sine curve is squashed, due to the larger range on the y-axis required to include the data from the file. We can fix that by plotting each curve against its own y-axis:

```
set y2tics -100, 20
set ytics nomirror
plot sin(x) axis x1y1 lw 3 title "Sine of x" , \
   "parabola.dat" axis x1y2 title "Parabola data"
```

Gnuplot can have two different y-axes and two different x-axes. In order to define a second y-axis, use the `y2tics` command; the first parameter is the starting value at the bottom of the graph, and the second is the interval between tics on the axis. The second line tells gnuplot to use a different axis on the right-hand side, rather than simply *mirroring* the left-hand y-axis. The final plot command is the same as the ones we've seen before, with the addition of the "axis" commands; these tell gnuplot which set of axes to use for which curve.

Notice how the previous graph doesn't really have room for the key anywhere inside it. One way to handle this is to put the key outside:

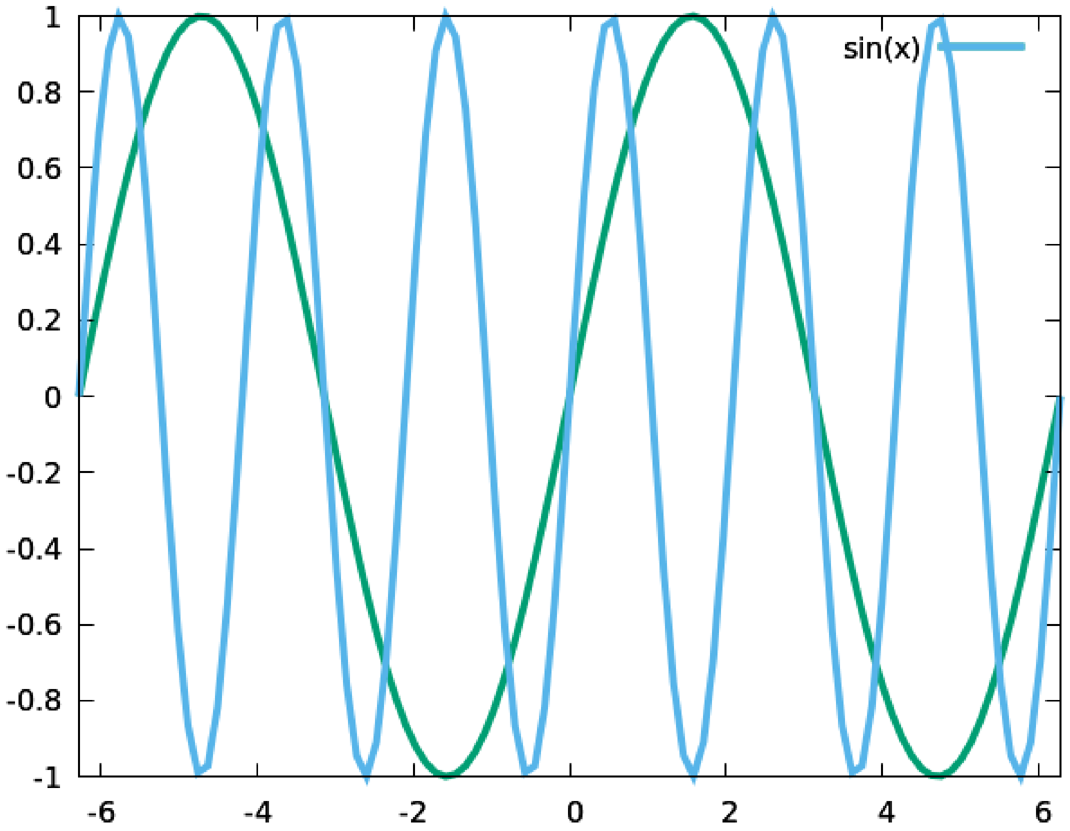
```
set key bottom left outside
plot sin(x) axis x1y1 lw 3 title "Sine of x" , \
    "parabola.dat" axis x1y2 title "Parabola data"
```



Multiplot

There is another way to get multiple plots on one graph, one that is ultimately more flexible. We'll explore the full capabilities of `multiplot` in a later chapter. For now, here is a simple example of how to use it to put two curves on a single graph, with each curve using its own settings. If you enter the commands one at a time at the terminal, notice how the prompt changes from >**gnuplot** to >**multiplot** after the `set multiplot` command is entered. To clear the graph and start over, you can enter another `multiplot` command.

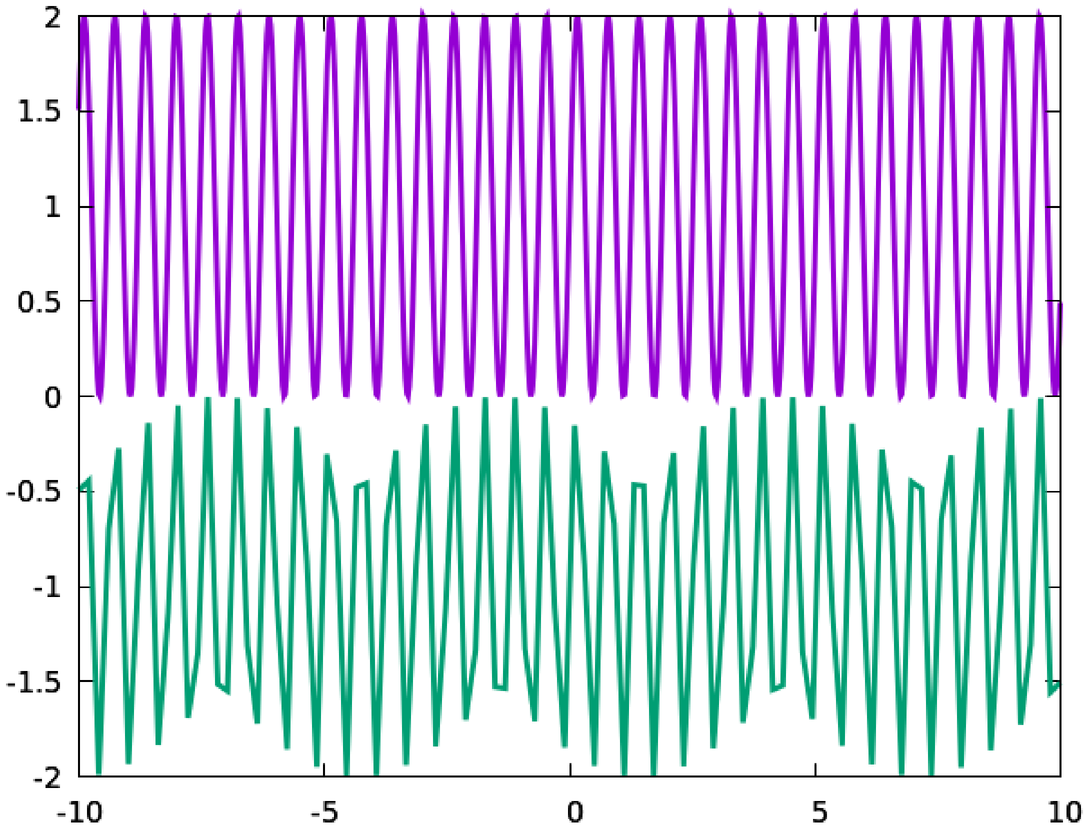
```
set multiplot
set xrange [-2*pi : 2*pi]
plot sin(x) lt 2 lw 4
set x2range [-6*pi : 6*pi]
plot sin(x) lt 3 lw 4 axis x2y1
```



Sampling Frequency

Gnuplot plots functions by dividing the x-axis into a number of points and evaluating the function at each point. These evaluation locations are called "samples." By default, gnuplot will use 100 samples. Sometimes this is not enough, and sometimes it is too many for our presentational strategy, as we'll see shortly. You can change the number of samples used with the `set samples` command. This will set the total number of (equally-spaced) sampled used for all the plot commands until you change it with another `set samples` command; but we can plot curves using different numbers of samples on the same graph by using the `multiplot` mode that we learned about in the previous example. When we reduce the number of samples, first our graph gets a little bumpy. When we reduce it too much, we see the effects of *aliasing*, and other symptoms of undersampling. Here is an example showing the same sine wave plotted twice. The two curves have different constants added to them to shift them vertically so we can see them both clearly on a single graph. Nothing else is changed except for the sampling frequency.

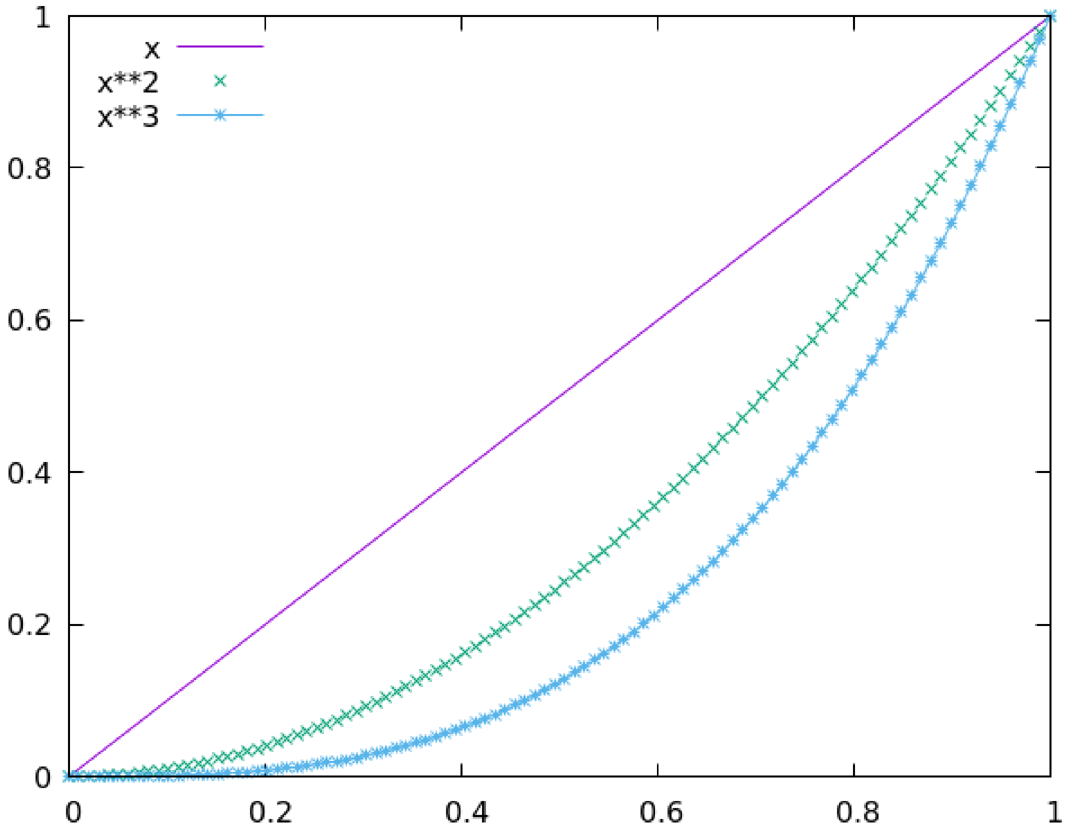
```
set multiplot
set nokey
set yrange [-2 : 2]
set samples 800
plot sin(10*x)+1 lw 3 lt 1
set samples 100
plot sin(10*x)-1 lw 3 lt 2
```



The “with” Command

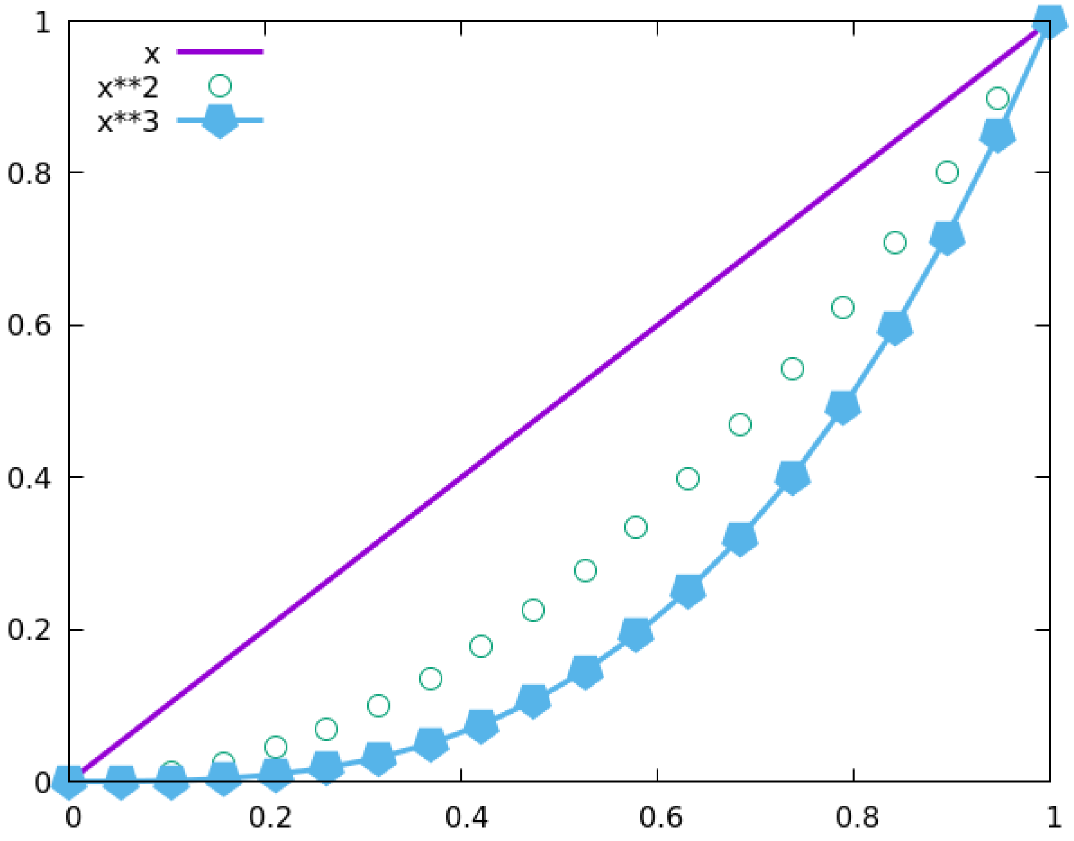
The three main styles for plotting functions or data are called `lines`, `points` (used for the parabola data in the previous two examples) and both together, called `linespoints`. The style can be chosen on the fly using the `with` command, as in the following example:

```
set key top left
set xrange [0 : 1]
plot x with lines, x**2 with points, x**3 with linespoints
```



Above we saw that the linetype can be selected from among those displayed using the `test` command. As you can see, the output of that command also displays examples of markers next to each sample line. You can have open or closed circles, diamonds, triangles, etc., but referring the `pointtype` (abbreviated `pt`). And just as you can set the line width with `lw`, you can set the point size with `ps`. This example, like all the others in this chapter, shows how to set styles on the fly, individually for each curve. In a later chapter we'll see how to define reusable styles and set styling defaults, to save typing and make our scripts easier to read. Gnuplot places a marker at each sample point. We've reduced the sampling frequency to make room for the larger markers.

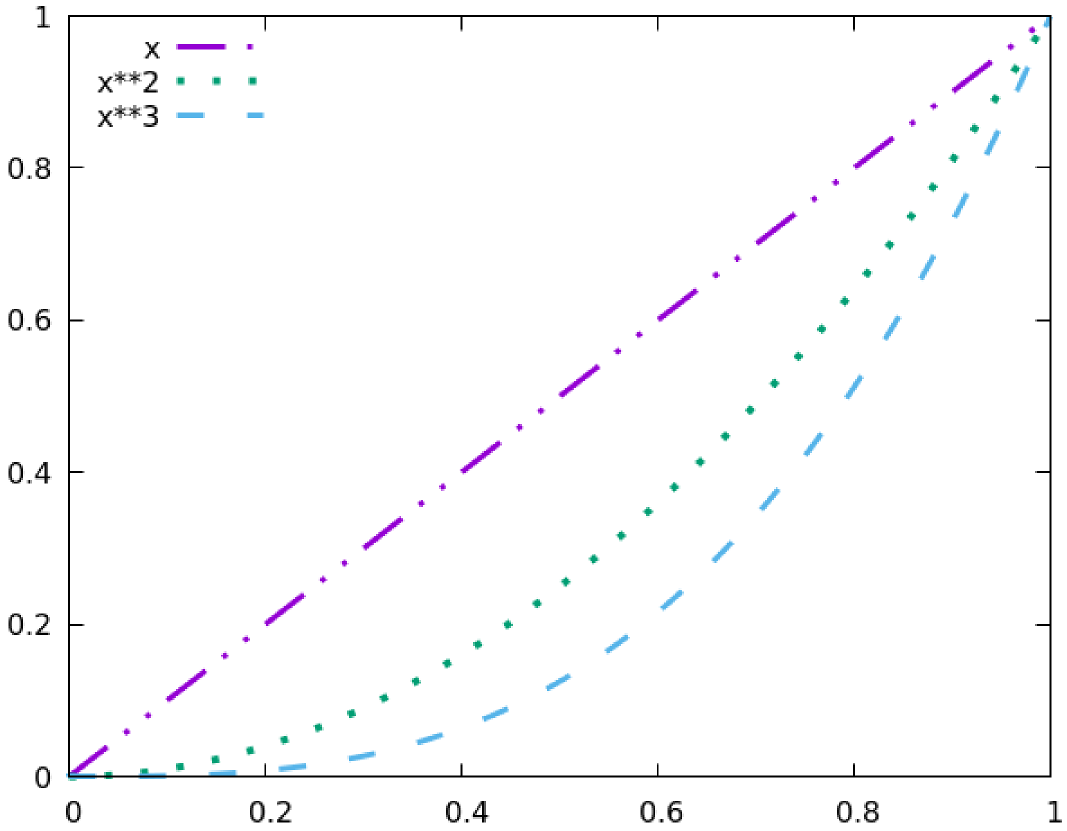
```
set samples 20
set key top left
set xrange [0 : 1]
plot x with lines lw 3, x**2 with points pt 6 ps 2,\
   x**3 with linespoints lw 3 pt 15 ps 3
```



Dashed Lines

You might have noticed, as well, that the output of the `test` command displays some dash patterns. You can set these as well, by specifying the `dashtype`, which can be abbreviated `dt`.

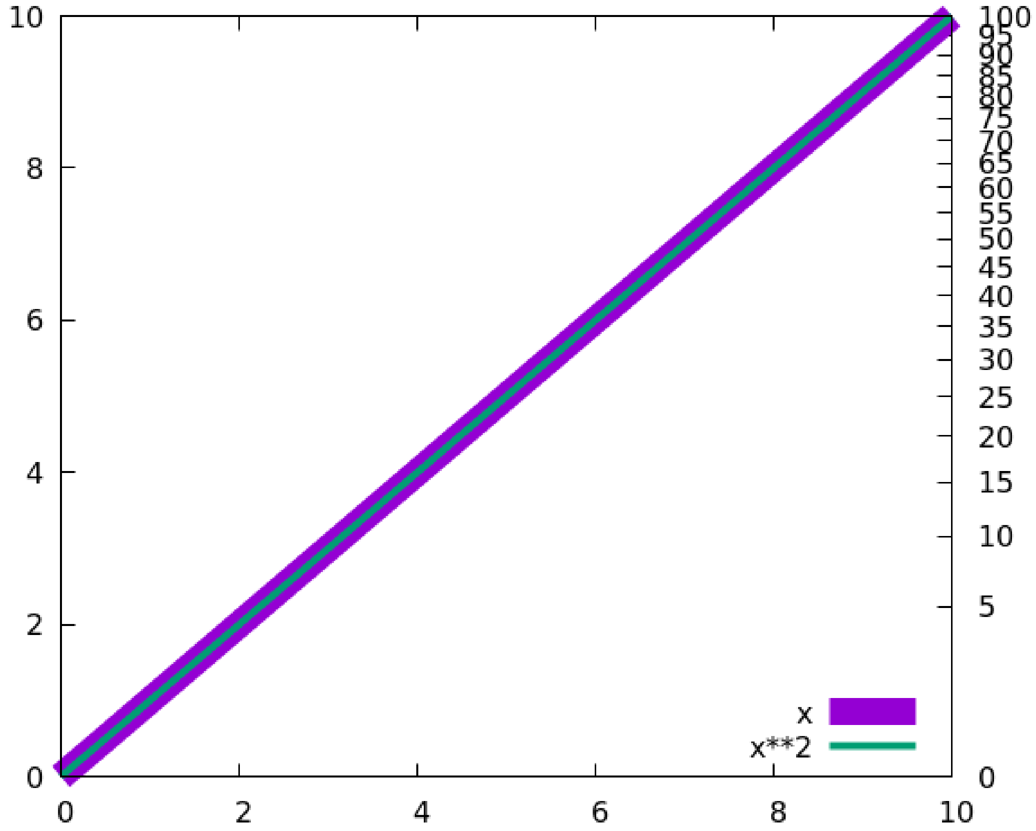
```
set key top left
set xrange [0 : 1]
plot x lw 3 dt 5, x**2 lw 4 dt 3,\
   x**3  lw 3 dt 2
```



The "set link" Command

Earlier in this chapter we learned how to set up a second y-axis so that two curves could be plotted together even though they covered very different ranges. The y2 axis can have its own range (set via `set y2range`) and its own tic spacing. Gnuplot also allows the two y-axes, or the two x-axes, to be related by any mathematical transformation. You do this by using the `set link` command, which requires you to spell out the inverse transformation as well:

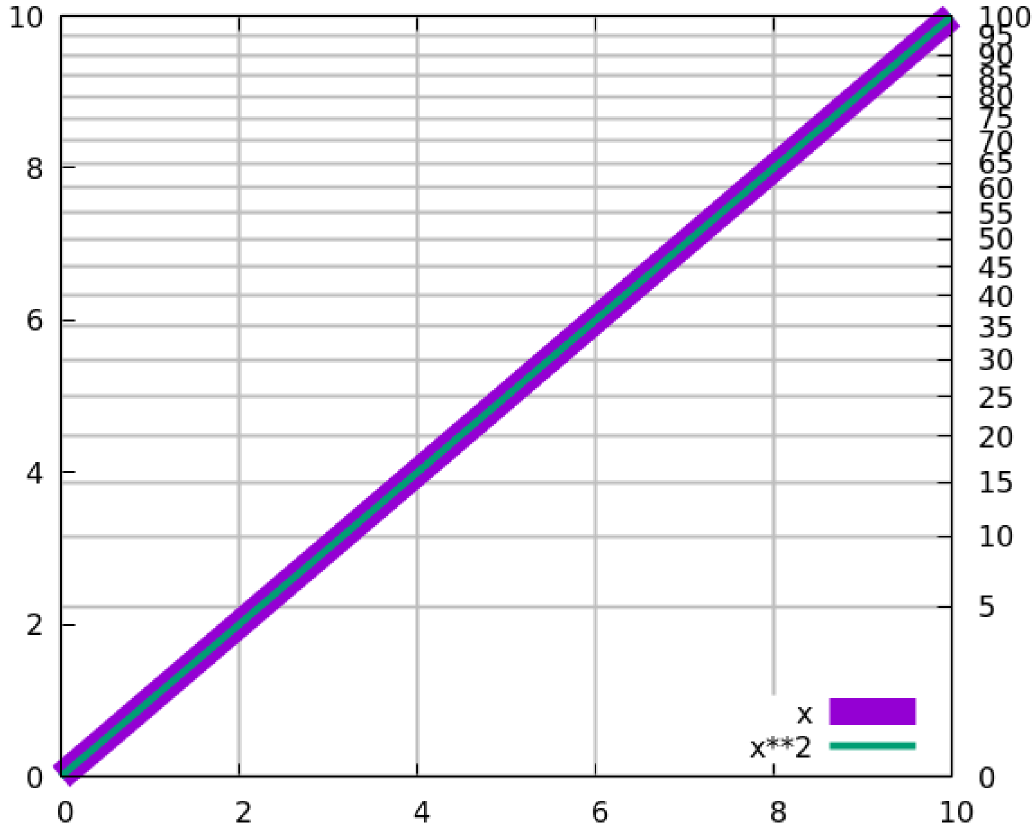
```
set key bottom right
set xrange [0 : 10]
set link y2 via y**2 inverse sqrt(y)
set ytics nomirror
set y2tics 0, 5
plot x lw 15, x**2 lw 4 axis x1y2
```



The previous example was designed to make it clear what the axis scaling does. As you can see from the ticmarks on the y2 axis, the scale on that axis is not linear, but is defined by the transformation in the `set link` command. We've plotted a straight line (the function "x") on the y1 axis, using a very thick width, and the function x^2 on the y2 axis, using a thinner line. The scaling of the y2 axis undoes the function, turning it into a straight line, which overlays the "real" straight line.

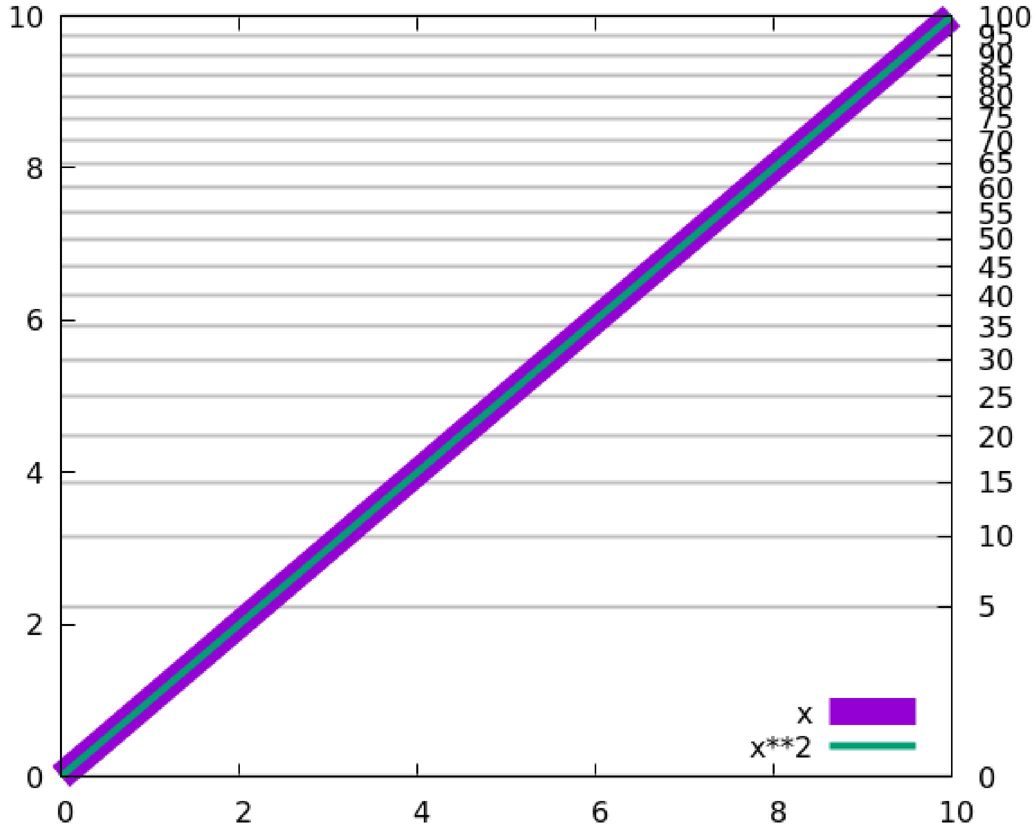
If we turn on the grid, using the `set grid` command that we learned about above, the grid will align with the y1 axis tics (and the x-axis tics). If we want it to align with the scaled, y2 axis, here's what we do:

```
set key bottom right
set xrange [0 : 10]
set link y2 via y**2 inverse sqrt(y)
set ytics nomirror
set y2tics 0, 5
set grid lw 2 lt 1 lc rgb "gray"
set grid noy y2
plot x lw 15, x**2 lw 4 axis x1y2
```



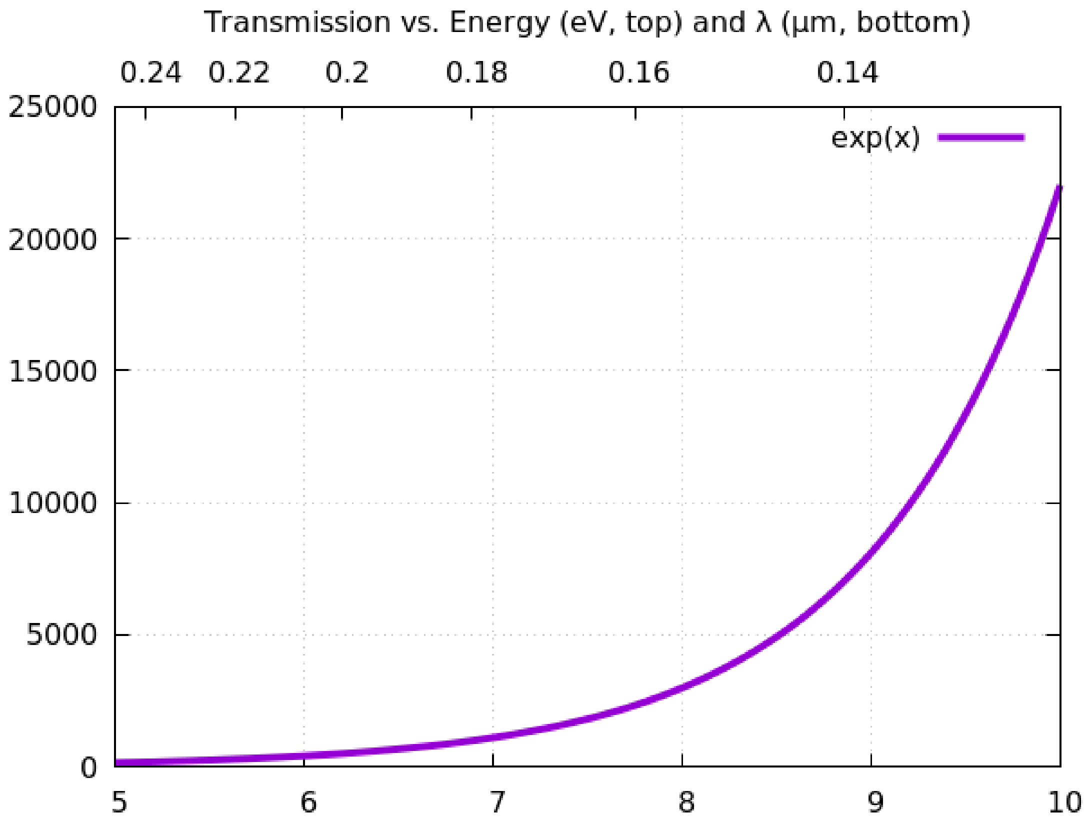
In the previous script, `y2` is acually an abbreviation for `y2tics`, etc. We had to specify `noy` (short for `noytics`) to turn off the grid for the first y-axis. Having them both on would lead to a confusing mess. You can also have a grid that extends the ticmarks of just one axis, by explicitly turning off the other one:

```
set key bottom right
set xrange [0 : 10]
set link y2 via y**2 inverse sqrt(y)
set ytics nomirror
set y2tics 0, 5
set grid lw 2 lt 1 lc rgb "gray"
set grid  y2
set grid nox noy y2
plot x lw 15, x**2 lw 4 axis x1y2
```



One very useful application of grid linking is showing different units on the same graph. Light can be talked about in terms of wavelength or of photon energy; the two are related by $E = hc/\lambda$, where E is the energy, h is Planck's constant, and λ is the wavelength. In the linking formula in the script below, we've used a multiplier that let's us express λ in microns and E in eV (electron volts). The highlighted line is the form of the `set xtics` command that sets in interval between the tics, letting gnuplot choose the start and end values automatically.

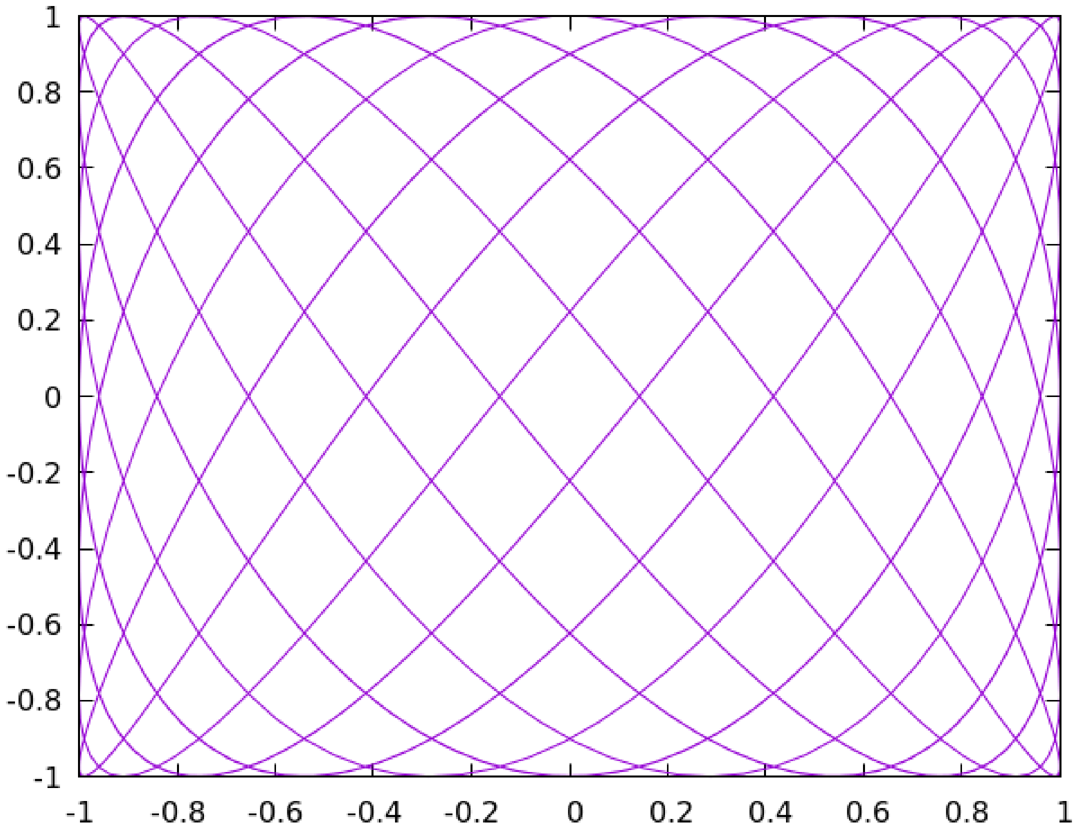
```
set title\
 "Transmission vs. Energy (eV, top) and λ (μm, bottom)"
set xrange [5 : 10]
set xtics nomirror
set link x2 via 1.24/x inverse 1.24/x
set x2tics .02
set grid
plot exp(x) lw 4
```



Parametric Plots

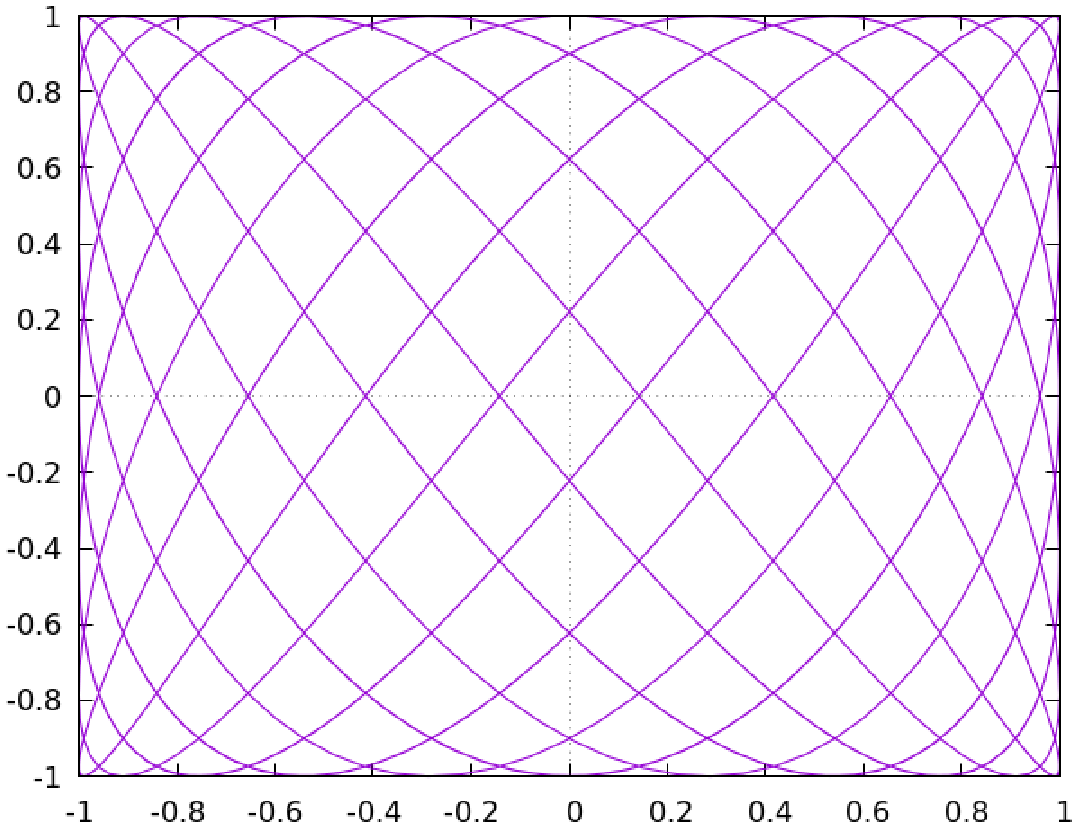
Up to now we've seen plots of functions and data where there was an explicit relationship between the x and y values. A more general class of 2D curves is where the x and y values each depend on a third variable, called a *parameter*. In gnuplot, the parameter is called "t". It has a default range, just as x does: [-5 : 5], and can be reset with the `set trange` command. The following plot resembles a Lissajous figure, which can be seen on an oscilloscope when sine waves of different frequencies are plugged into the x and y axes.

```
set samples 1000
set parametric
unset key
plot sin(7*t), cos(11*t)
```



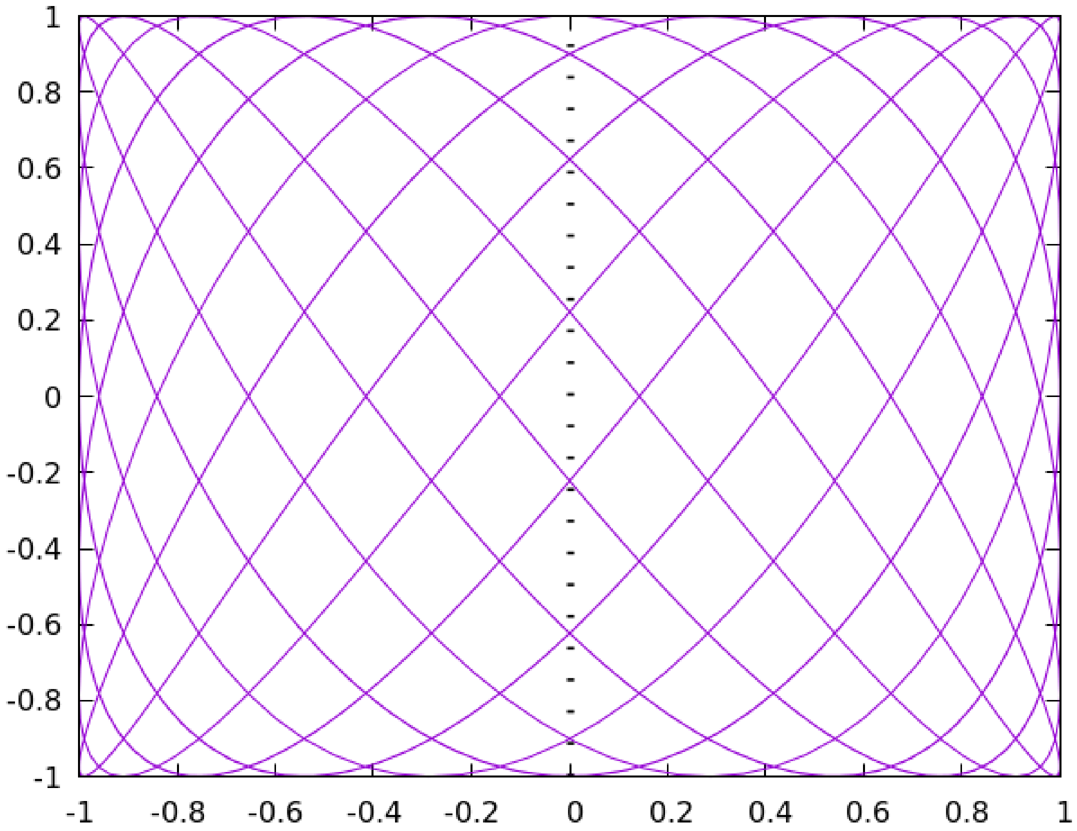
Up to now the tic marks and labels have been placed around the outside of the plot, at what gnuplot calls the “border”. This is not where the actual axes are. Unless you turn them on, the axes are not drawn.

```
set samples 1000
set parametric
unset key
set zeroaxis
plot sin(7*t), cos(11*t)
```



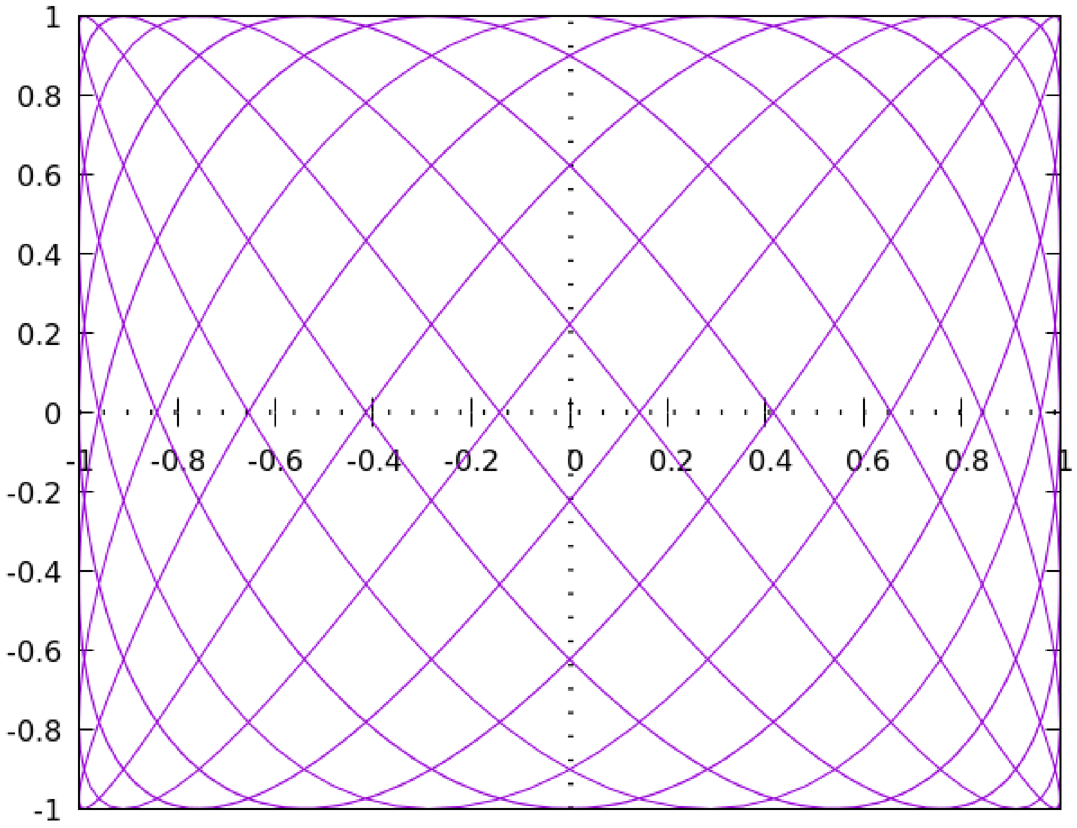
You can choose to show only one of the axes. Let's try that while also showing how to set the thickness of the axis line. Linetypes and colors can be set as well.

```
set samples 1000
set parametric
unset key
set yzeroaxis lw 4
plot sin(7*t), cos(11*t)
```



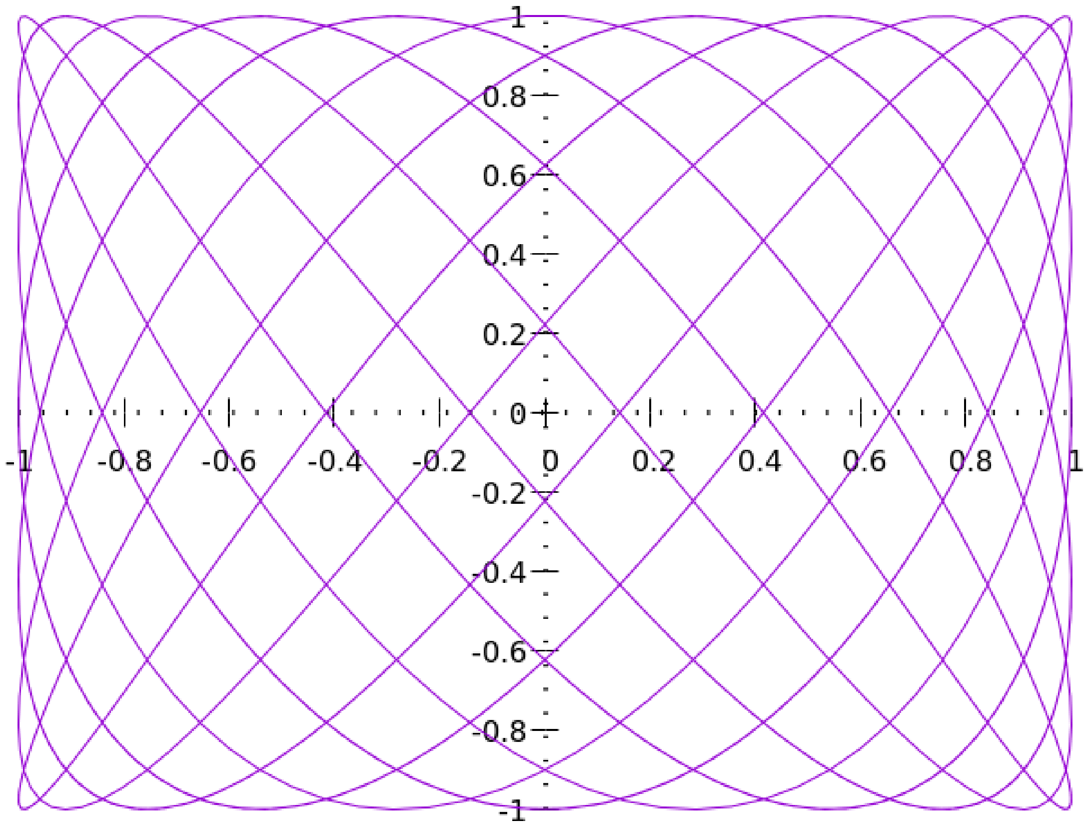
In the previous examples the ticmarks and their numerical labels stayed on the border. We can move the x-tics, y-tics, or both to the axes.

```
set samples 1000
set parametric
unset key
set zeroaxis lw 3
set xtics axis
plot sin(7*t), cos(11*t)
```



Here we move all the tics to the axes and dispense with the border entirely:

```
set samples 1000
set parametric
unset key
set zeroaxis lw 3
unset border
set xtics axis
set ytics axis
plot sin(7*t), cos(11*t)
```



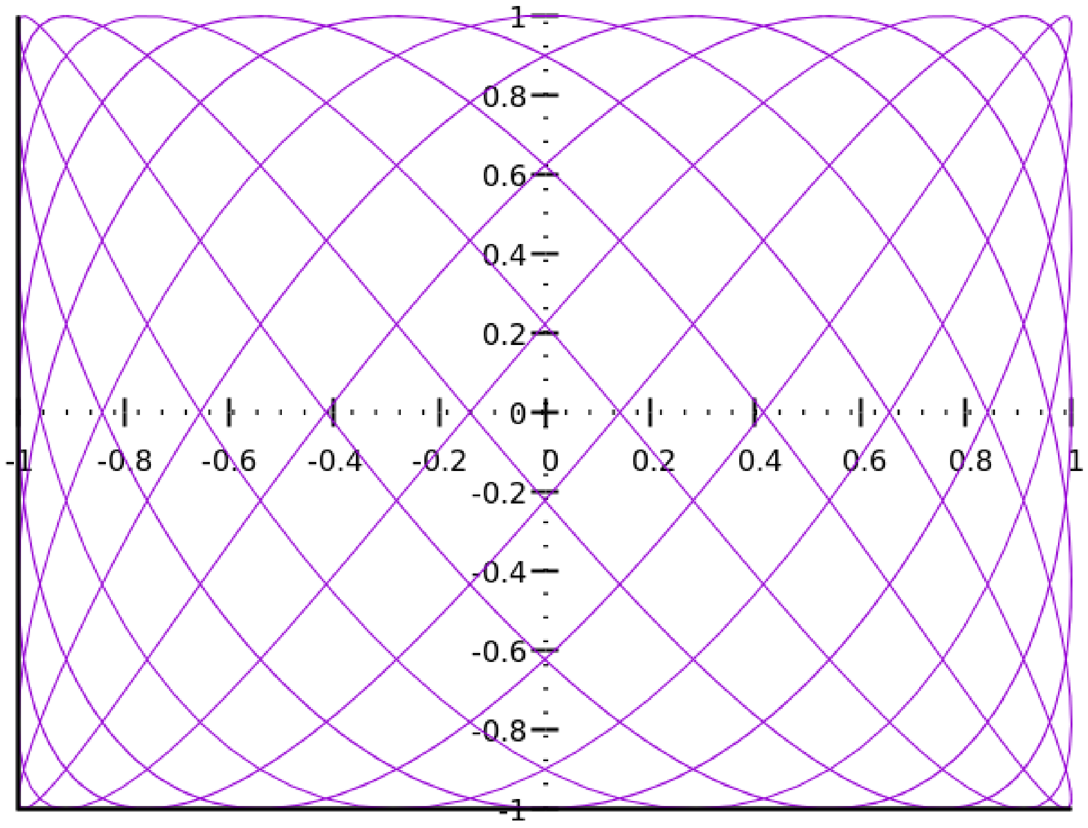
Controlling Your Borders

You can have a partial border: on the top, left, or any combination. Specify where by adding these numbers for the segments that you want, and using the result as a parameter in the `set border` command:

```
1       bottom
2       left
4       top
8       right
```

So to get (horizontal) borders on the top and bottom only, the magic number is 5. Here's how to use this to get borders on the bottom and left:

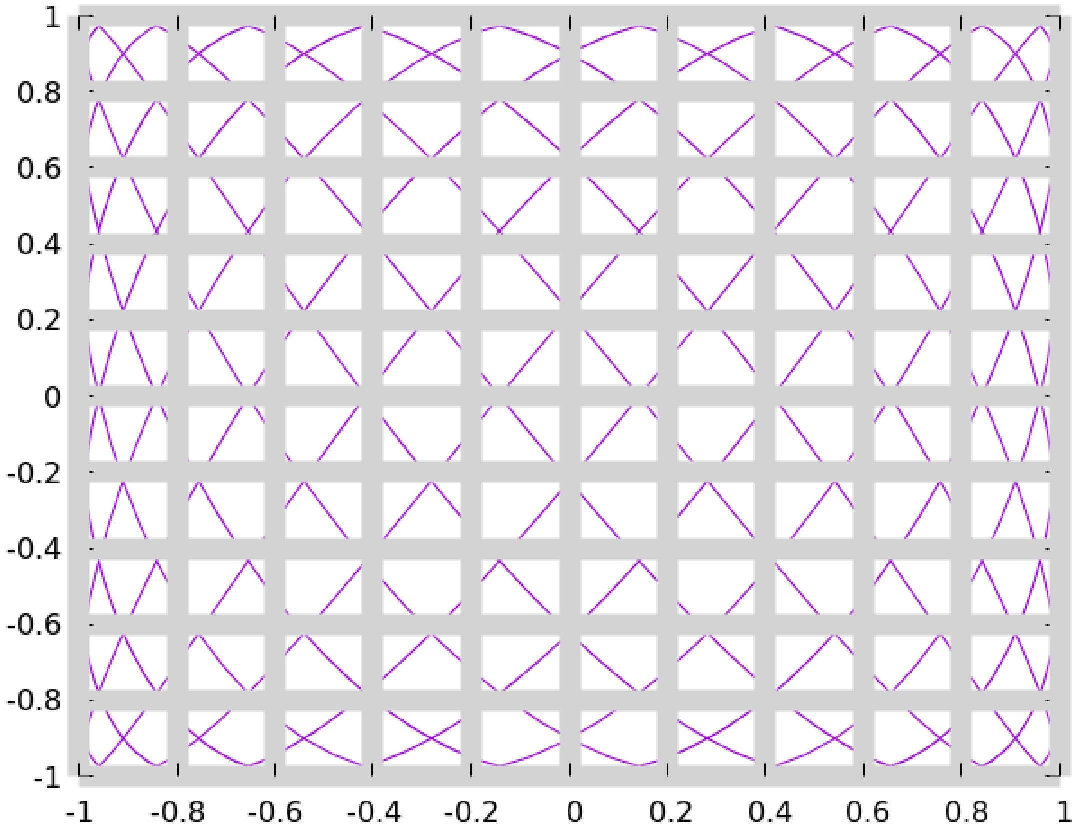
```
set samples 1000
set parametric
unset key
set zeroaxis lw 3
set border 3 lw 2
set xtics axis
set ytics axis
plot sin(7*t), cos(11*t)
```



Front and Back

Borders and grids can be drawn in front of or behind the data. Unless you reset it, the grid is drawn behind the curves and the border in front. We'll illustrate the effect of changing the default with some thick gridlines. To review the meanings of the abbreviations in the second to last line of the script below: `lw 12` means 12 times the terminal's default linewidth; `lt 1` means linetype 1, usually a solid line; and `lc` is short for linecolor, where we've chosen one of the convenient color names. Along with the "front" keyword, there is a "back" keyword that does what you would expect.

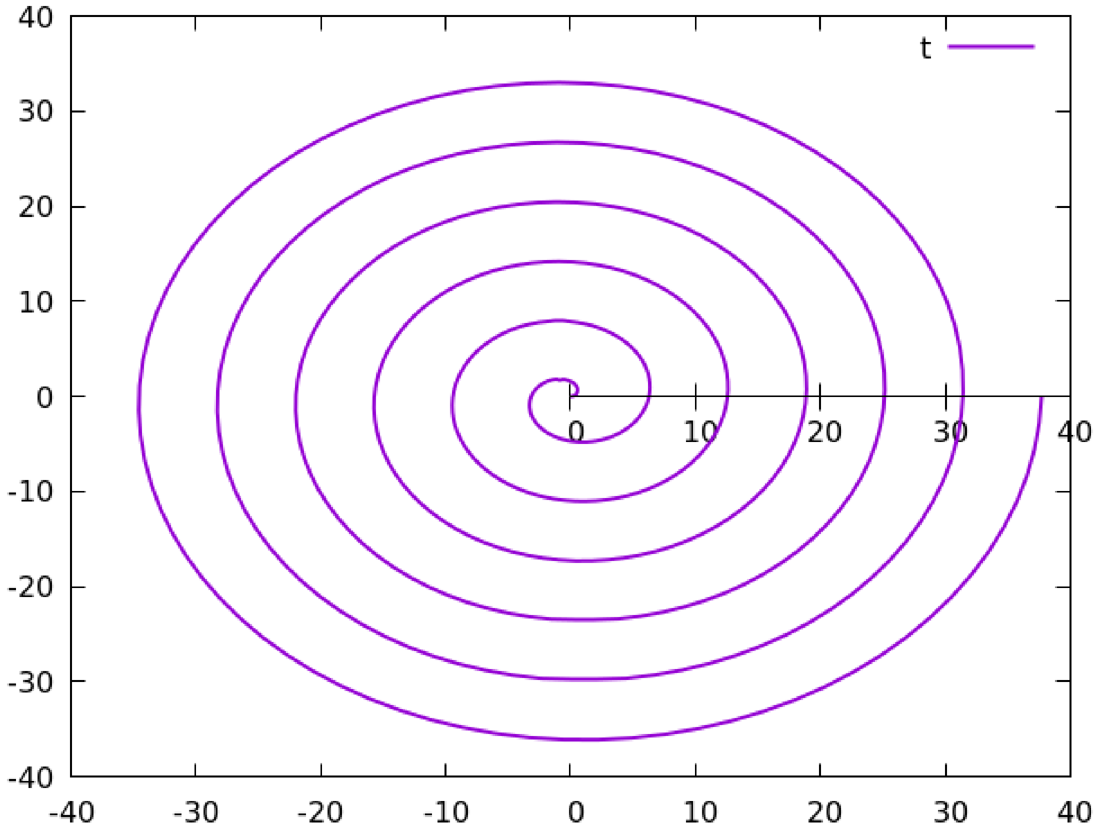
```
set samples 1000
set parametric
unset key
unset border
set grid lw 12 lt 1 lc rgb "light-gray" front
plot sin(7*t), cos(11*t)
```



Polar Coordinates

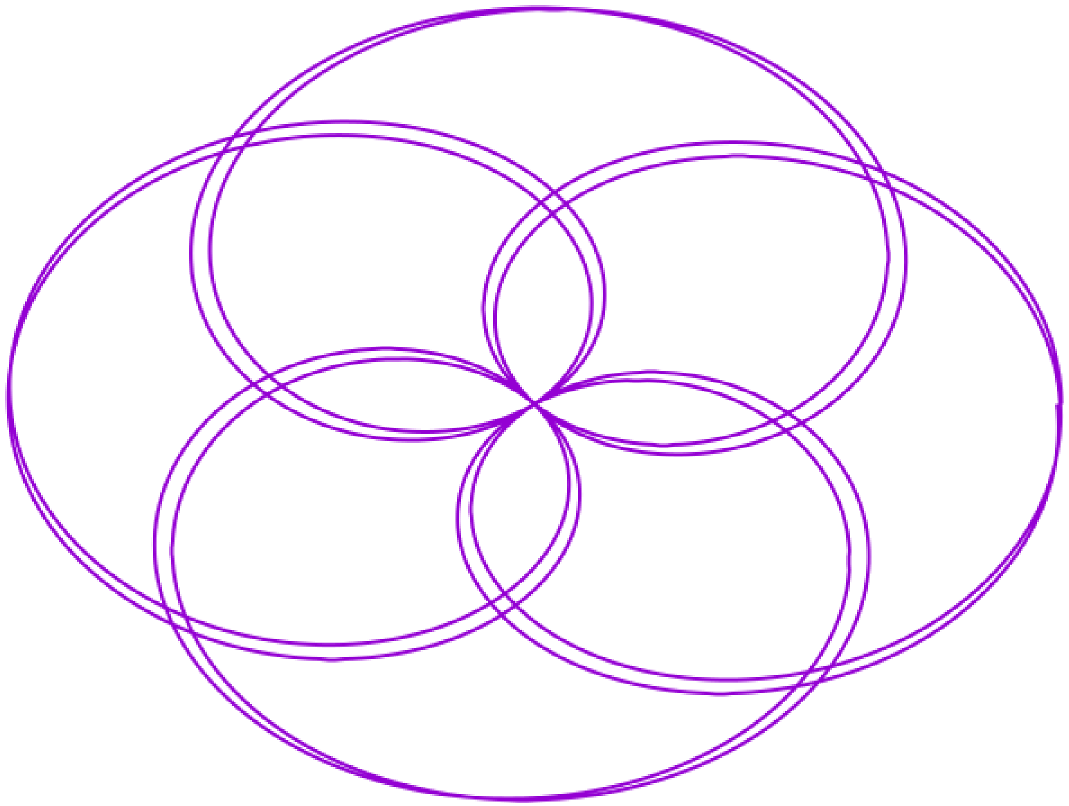
All the plots in this chapter up to now have used rectangular coordinates, which gnuplot calls, following the usual convention, x and y. For certain types of situations, however, *polar* geometry is the natural coordinate system. In polar coordinates we have a radius, r, measured from the origin (which is usually at the center of the graph) and an angle, θ, usually measured counter-clockwise from the horizontal. On the gnuplot command line, the angular coordinate is called "t", and has a default range of 0 to 2π; for this plot we want to cover a larger range of angles, so we set the `trange` accordingly. This example also demonstrates gnuplot's default treatment of the axes in a polar plot, adding a radius axis to the usual borders.

```
set samples 500
set polar
set trange [0 : 12*pi]
plot t lw 2
```



The previous graph depicts the function r = t, which we got just by telling gnuplot to `plot t`. This is sometimes called Archimedes' spiral. It is analogous to plotting a straight line in rectangular coordinates, or the function y = x, by telling gnuplot to `plot x`. Using polar coordinates, we can easily generate some complicated looking plots, such as the spirograph-type curve shown here. In this example we've turned off all the tics and axes, including those associated with the radius coordinate, leaving just a decorative curve.

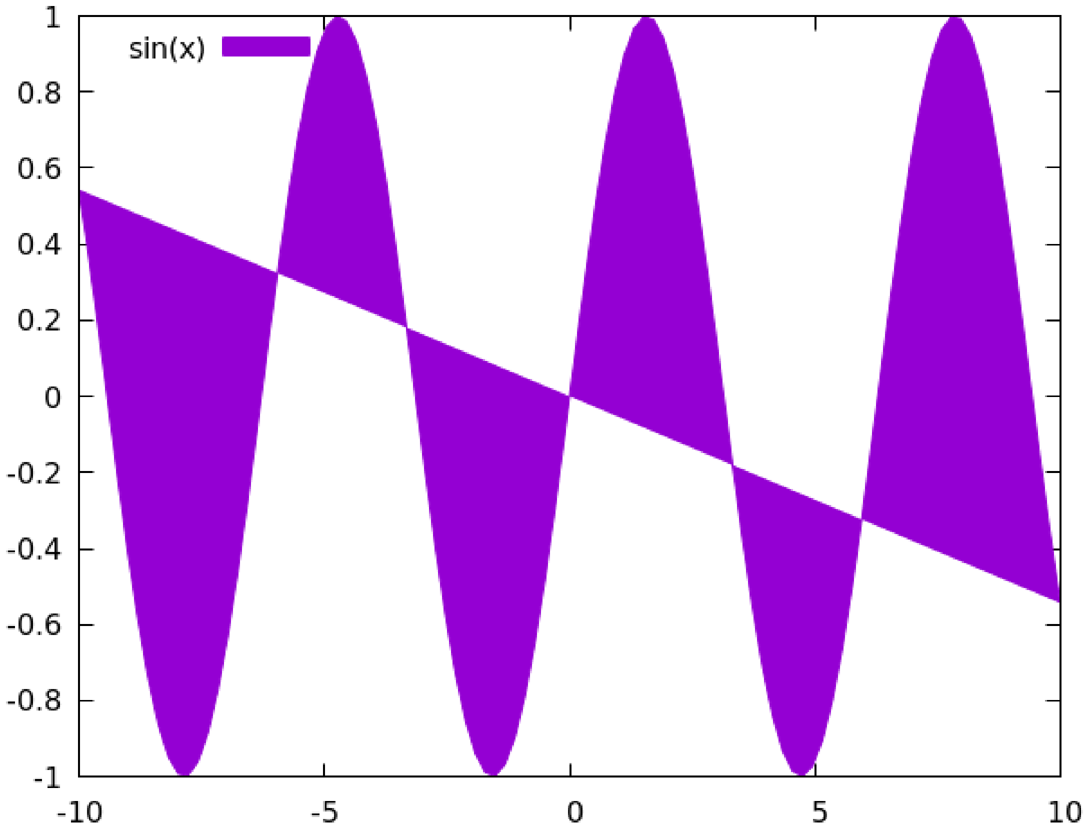
```
set samples 2000
unset key
unset border
unset xtics
unset ytics
unset rtics
unset raxis
set polar
set trange [0 : 12*pi]
plot cos(0.67*t) lw 2
```



Filled Curves

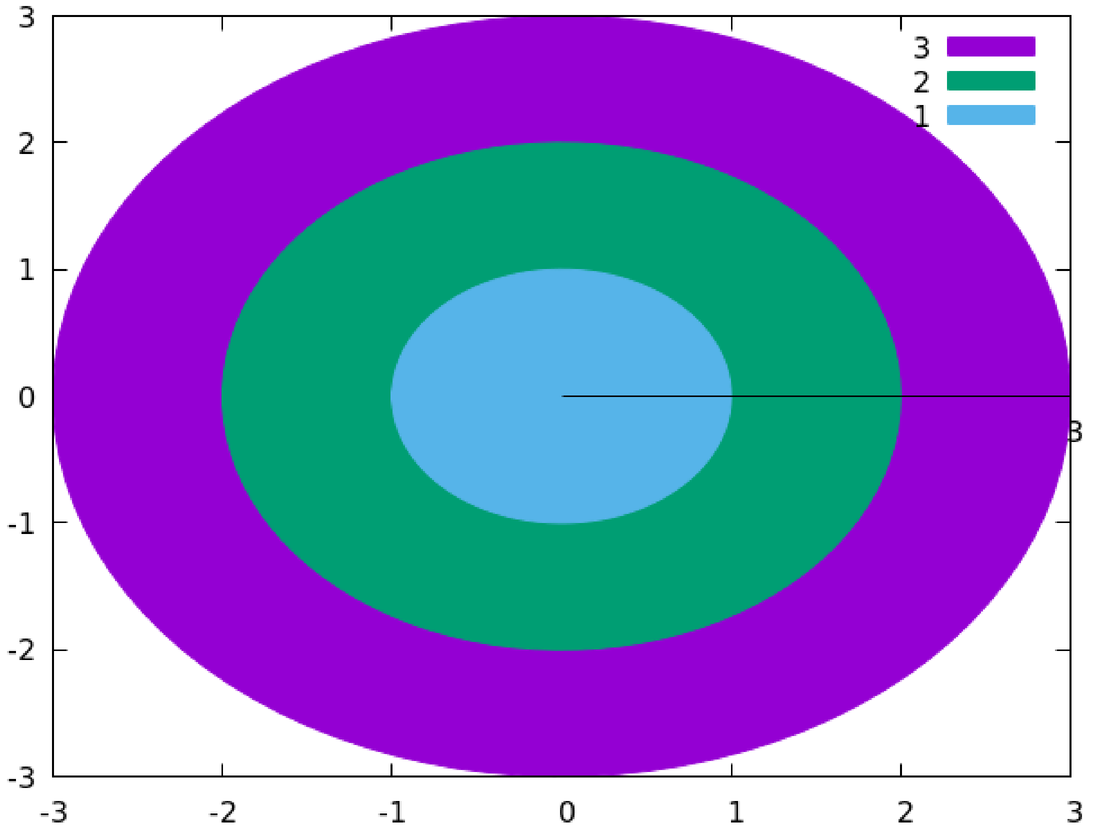
Gnuplot can *fill* portions of the curves you plot with colors or patterns. There are a handful of options for filling curves; we'll give an example of each one. Here is the default if you just use the `with filledcurves` command with no options. It treats the curve as if it were closed and fills up the resulting area:

```
set key top left
plot sin(x) with filledcurves
```



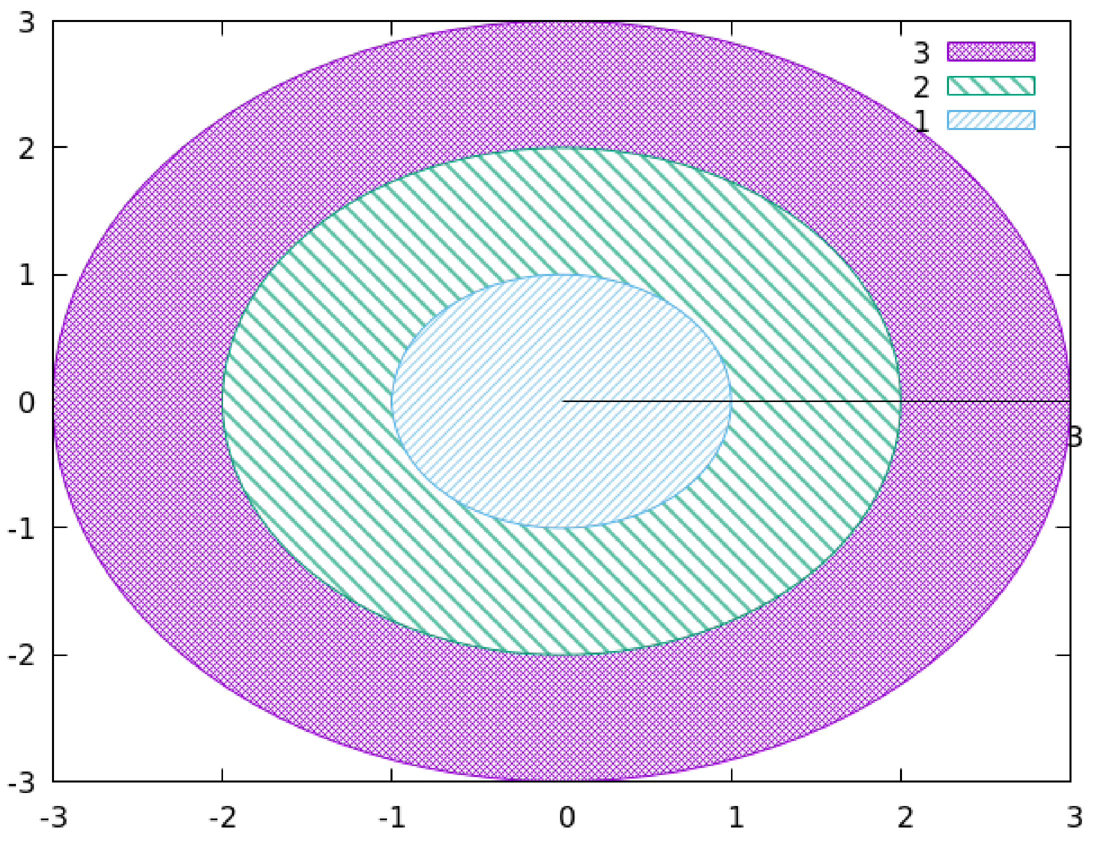
In the previous example the program just closed the curve by drawing an imaginary line from the final point plotted to the first point. This default behavior is perhaps more useful in certain polar plots, like the one we show here. Note the order in which the plot commands are issued; this is the order in which gnuplot will draw the curves and paint the fills. It should be clear that when we say `plot 3` in polar coordinates that we are asking for a circle with radius = 3; because the set of points for which the radius is a fixed value is the definition of a circle. When you try this example, you may find that, instead of circles, gnuplot is drawing what look like ellipses. This is because it is setting up the plots on your terminal to cover a non-square rectangular area. This usually looks better than a square plot, but if you want your circles to look like circles, add the command `set size ratio 1` before the `plot` command. We'll have more to say about sizing and positioning plots in a later chapter.

```
set polar
plot 3 with filledcurves, 2 with filledcurves,\
   1 with filledcurves
```



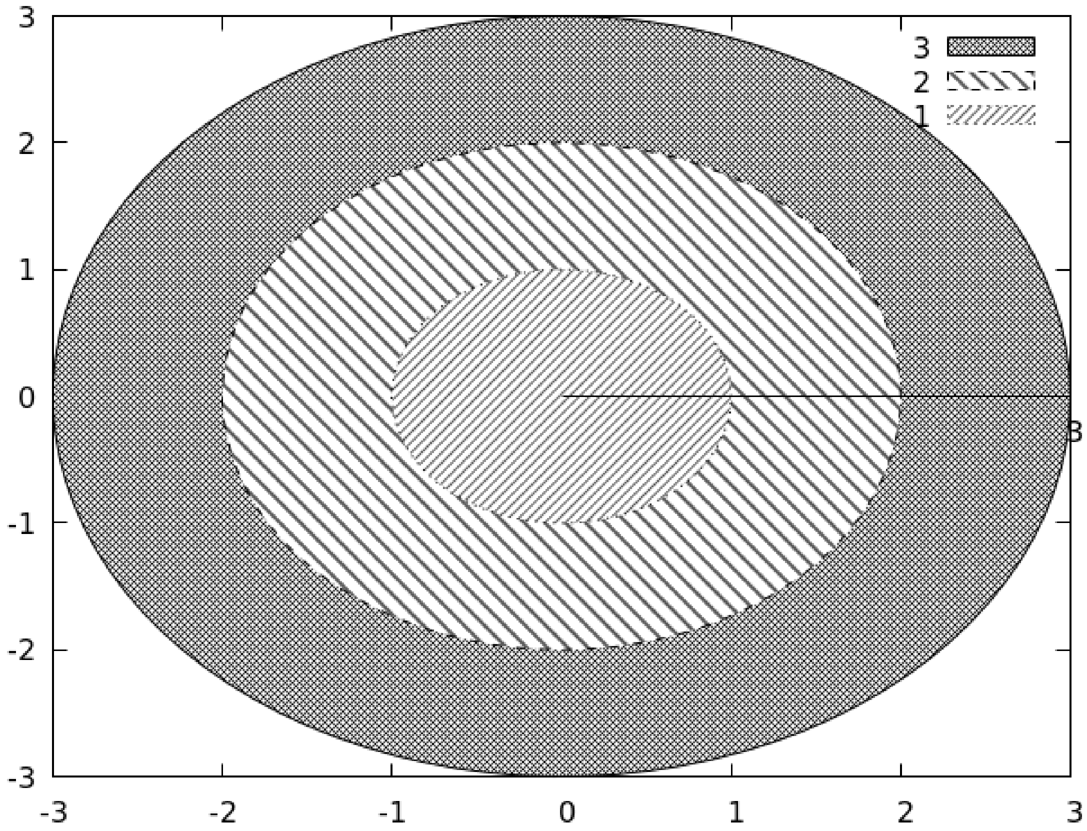
In the previous example, we let gnuplot create the fills with its default sequence of solid colors. These can be altered using the `with fillcolor` command, or its abbreviation, `with fc`, just as linecolors can be set with the `with lc` command. A later chapter will be devoted entirely to color in gnuplot; there we'll learn how to do such things as overlaying fills with different colors and opacities. But we have another option. If you look at the output of the `test` command again, you will notice a collection of "pattern fill"s. These depend on the terminal in use. Here is how to use them to fill areas with patterns rather than solid colors. This is particularly useful when preparing plots for publication when color is not an option. In the code below, `fs` is an abbreviation for `fillstyle`.

```
set polar
plot 3 with filledcurves fs pattern 2,\
   2 with filledcurves fs pattern 4,\
   1 with filledcurves fs pattern 7
```



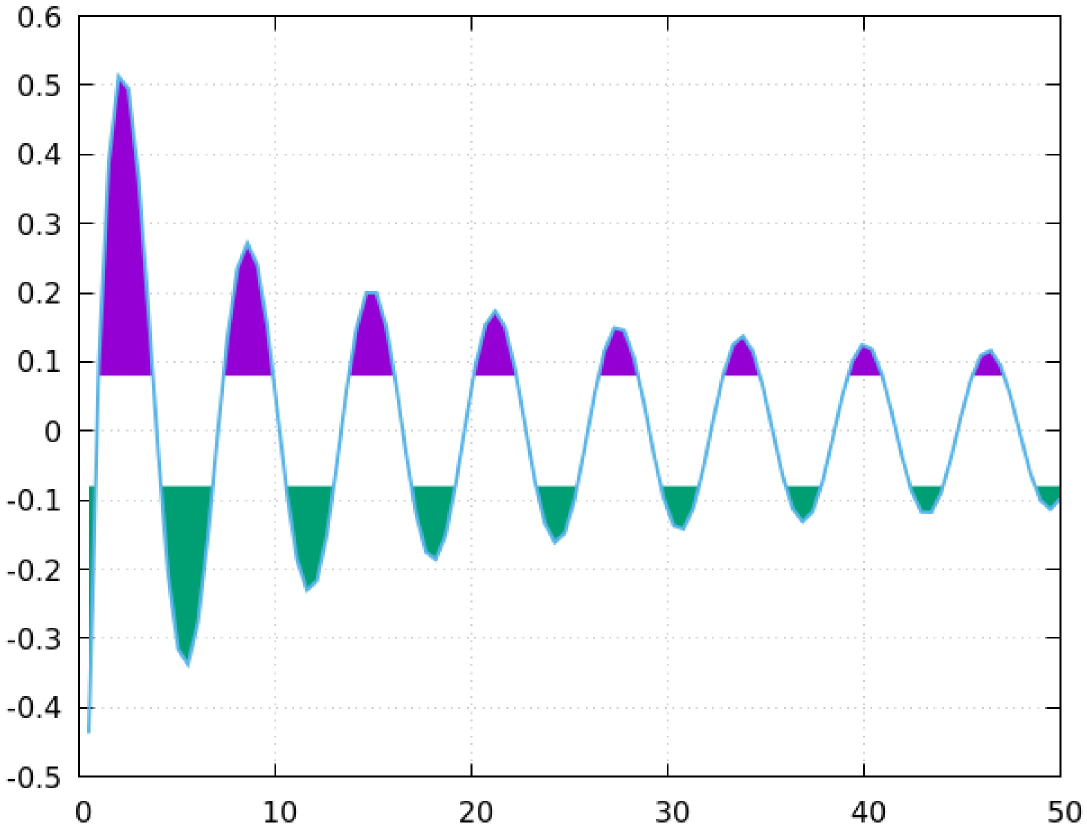
In the previous example of pattern fills, gnuplot used the patterns we specified, but also applied a sequence of colors. If you want a strictly monochrome rendering, you can combine patterns with the command for that:

```
set polar
set monochrome
plot 3 with filledcurves fs pattern 2,\
     2 with filledcurves fs pattern 4,\
     1 with filledcurves fs pattern 7
```



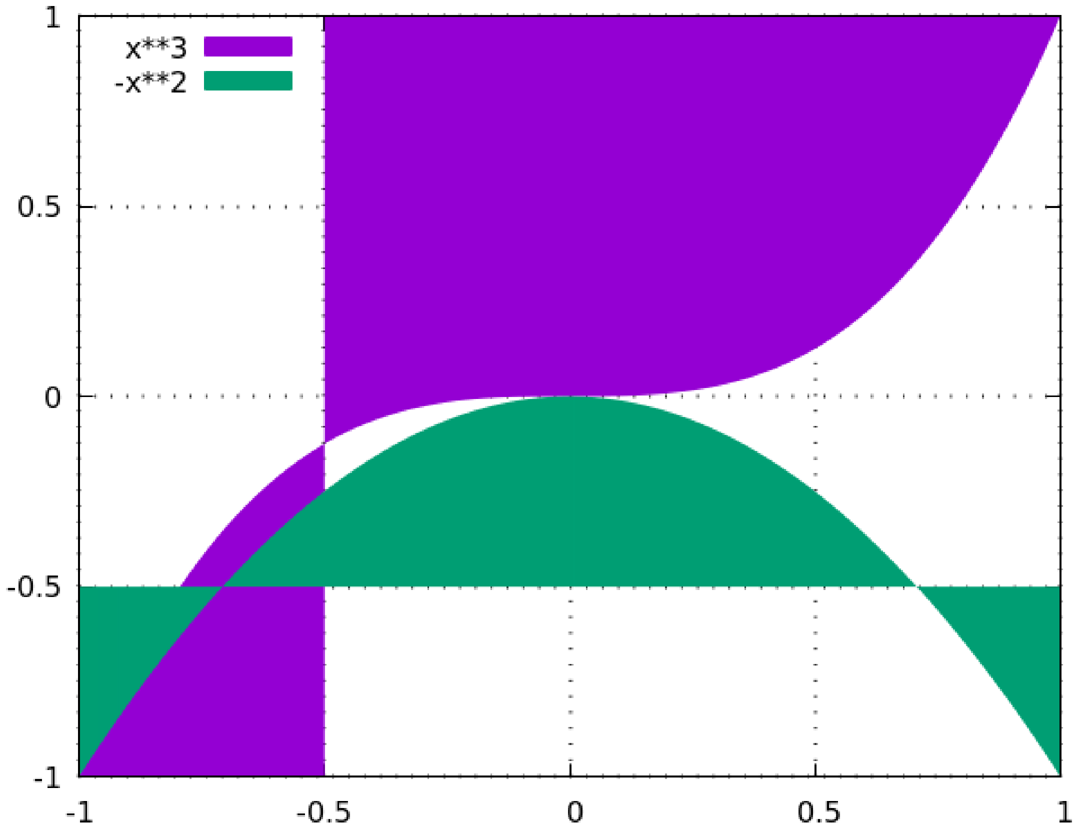
Now let's return to rectangular coordinates to illustrate the other filledcurves options. Sometimes you want to fill in the part of a curve that lies above or below a particular value. Here is an example showing how to do both on one graph, using a Bessel function (to get a list of the other special functions built-in to gnuplot, issue the command `help expressions functions`). After doing the filledcurves plots, we need to plot the curve itself, as the filledcurves plots just plot the fills:

```
unset key
set grid
set xrange [0 : 50]
plot besy0(x) with filledcurves above y = 0.08,\
     besy0(x) with filledcurves below y = -0.08,\
     besy0(x) lw 2
```



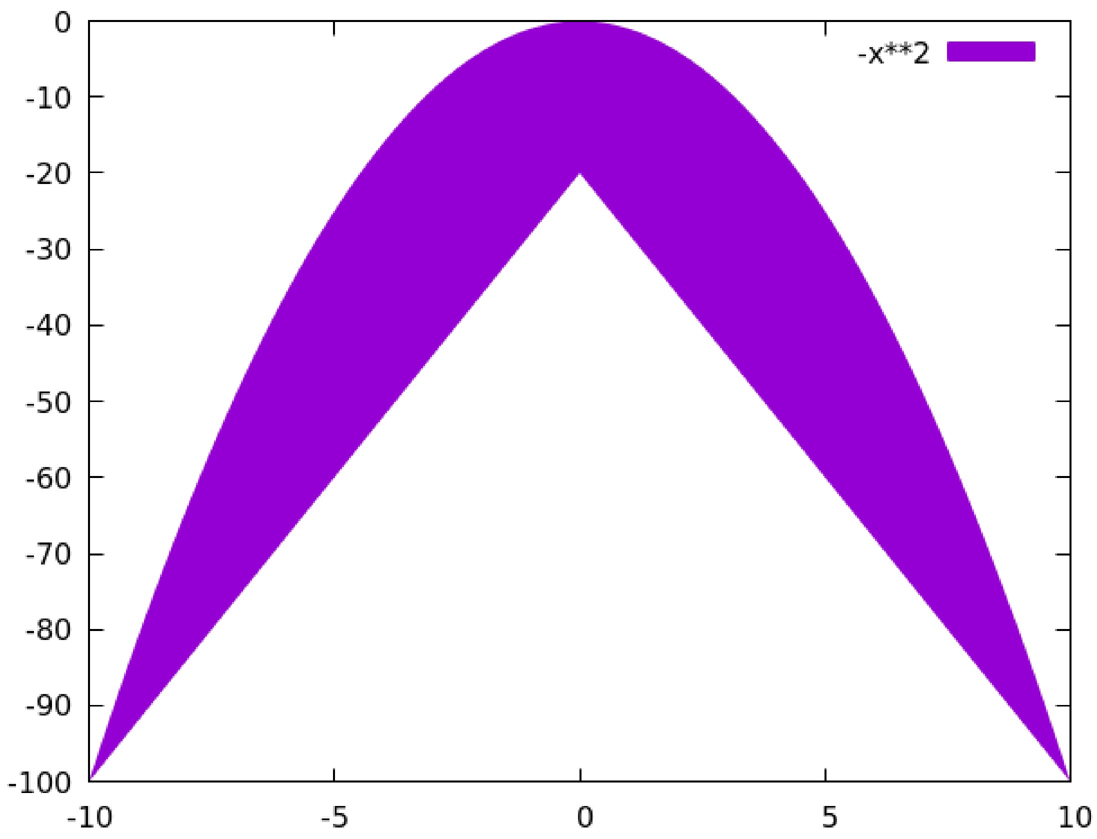
Another option is to fill the area between the curve and a vertical or horizontal line:

```
set key top left
set xrange [-1 : 1]
set grid lw 2
plot x**3 with filledcurves x=-0.5,\
    -x**2 with filledcurves y=-0.5
```



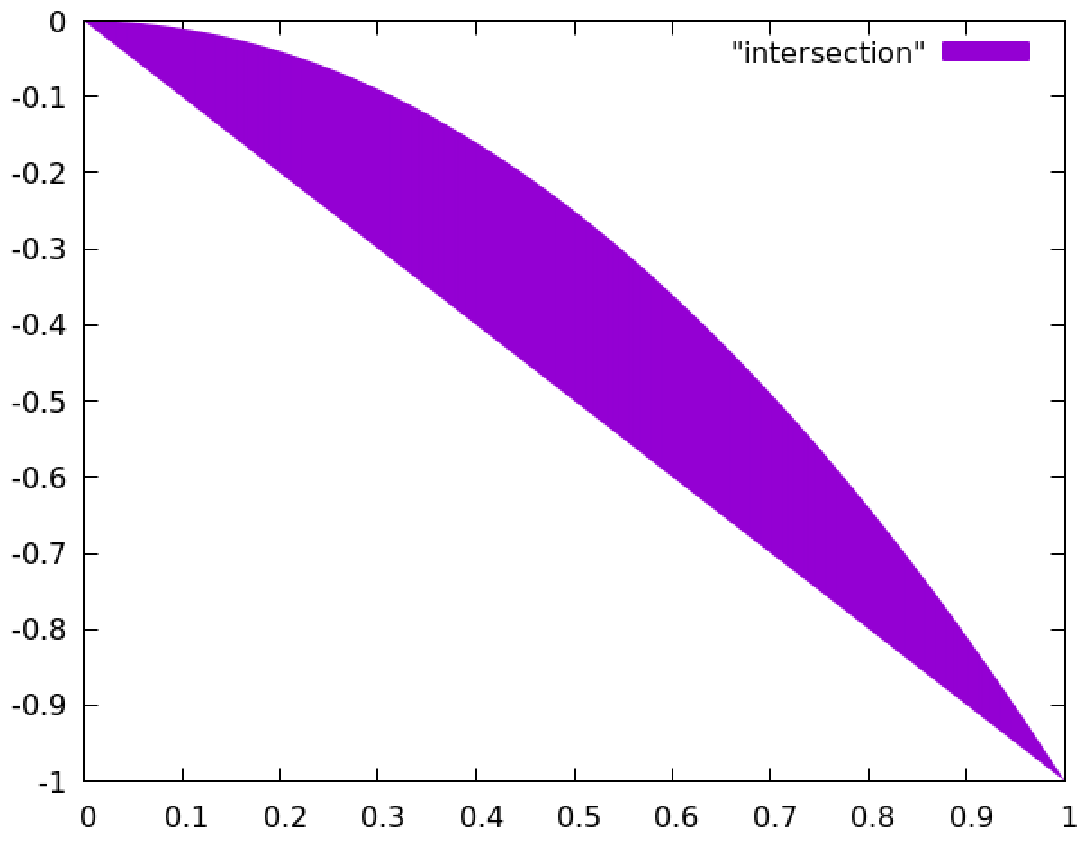
You can also fill the curve between a point, given in x,y plot coordinates, and the end points of the curve:

```
plot -x**2 with filledcurves xy = 0, -20
```



The final filledcurves option is to fill in the area between two curves. This is a bit more complicated, as it requires a data file, rather than functions specified on the command line. The data file must have rows of the form *x y1 y2*; the area between the curves y1 and y2 will be filled. You can make your own data file or use the file called "intersection" that we've made available with this book. That file contains the coordinates of a straight line with a negative slope and a downward-opening parabola. If you start up gnuplot in the same directory where you have stored the file "intersection" you just need the one line in the script below to make a plot showing the area between the two curves. In later chapters we'll learn how to make other types of plots from the same file, and a trick to make this plot with no data file at all.

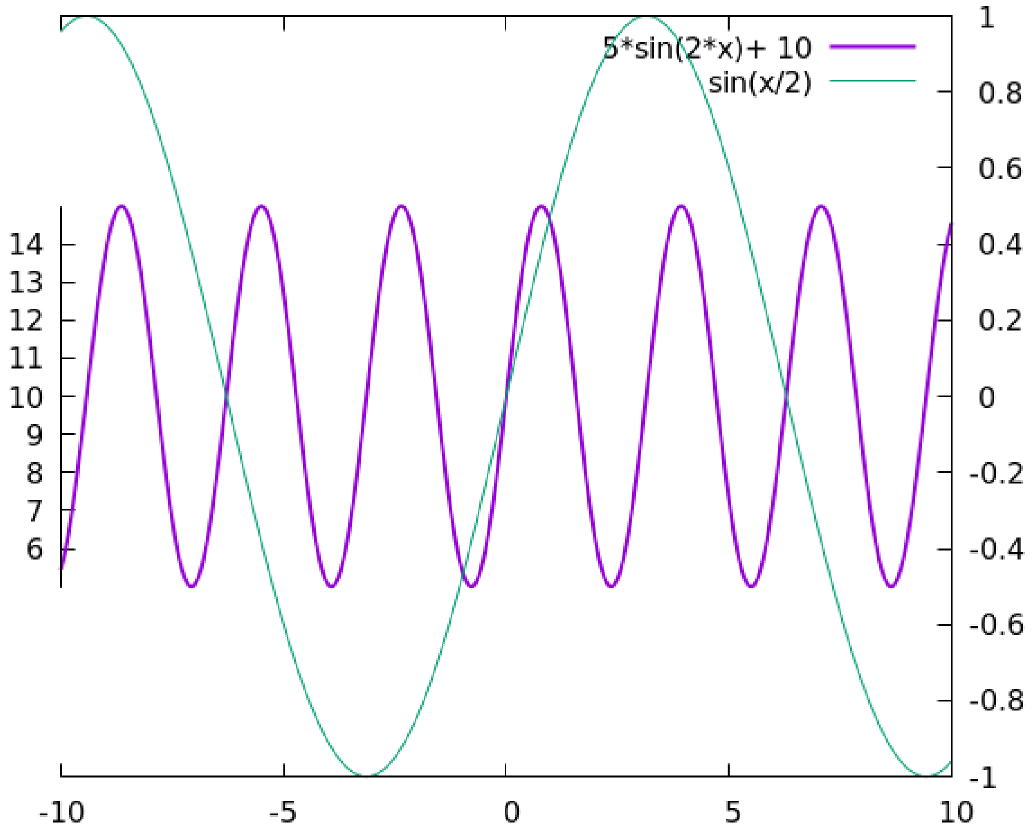
```
plot "intersection" with filledcurves
```



Range-frame Graphs

We'll round out this chapter with a few examples illustrating some variations on range handling in gnuplot. Up to now we've relied on the set xrange and related commands for setting the maximum and minimum values on the various axes. Gnuplot can also produce what are sometimes called "range-frame" graphs, where an axis and its tics are limited to the data actually plotted, even if the graph as a whole may cover a larger range. An example should make this clearer:

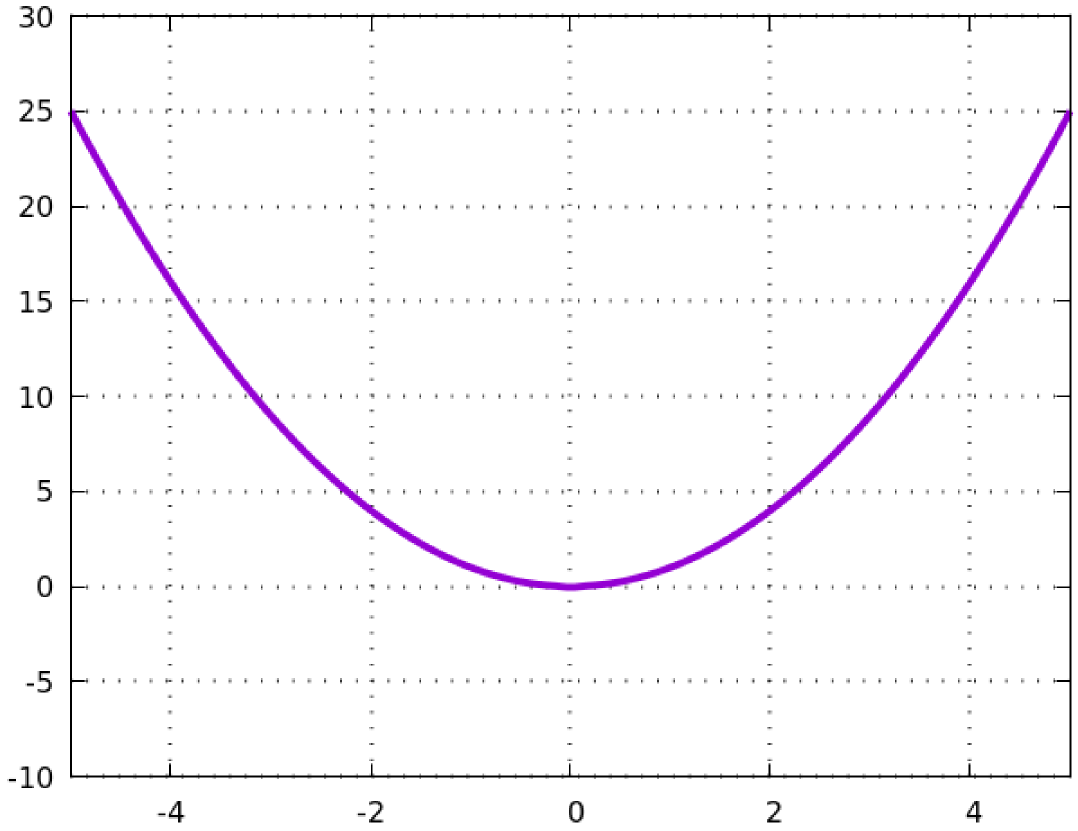
```
set samples 1000
set ytics nomirror rangelimited
set ytics 1
set y2tics 0.2
set y2range [-1:1]
set yrange [0:20]
plot  5*sin(2*x)+ 10 lw 2, sin(x/2) axis x1y2
```



Local Ranges

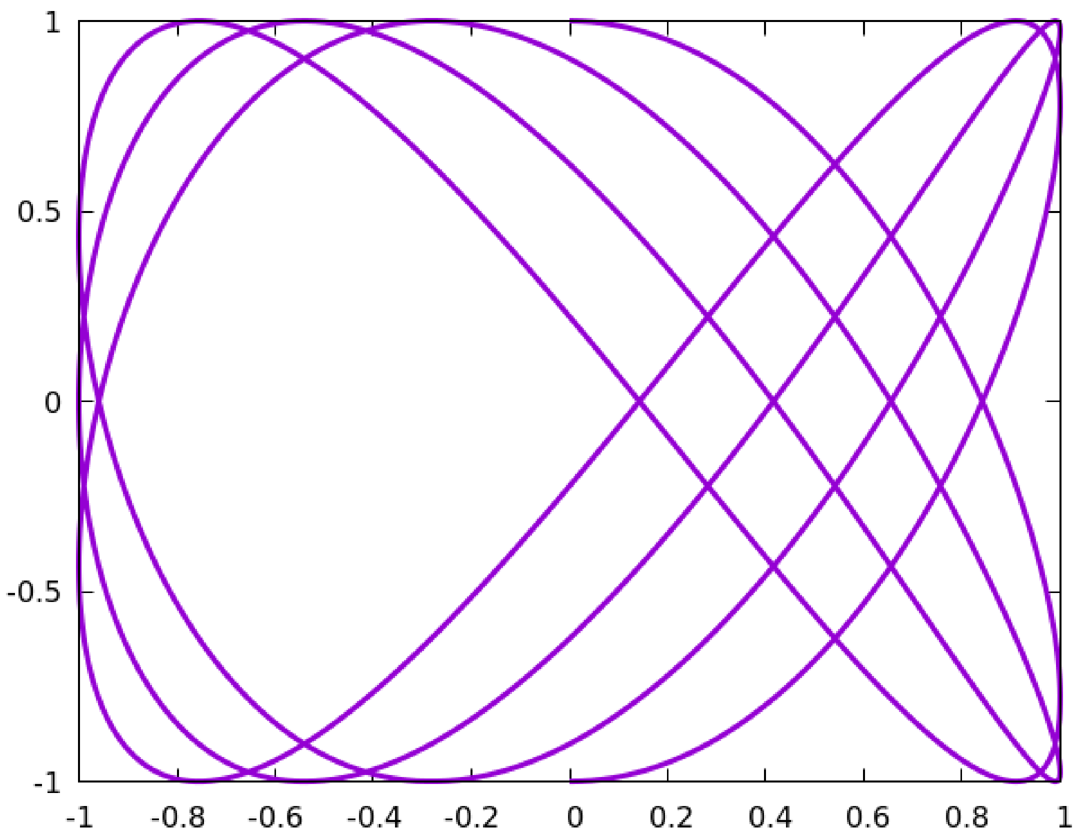
The `set yrange` and related commands set the ranges globally, applying to all subsequent plot commands. Gnuplot also allows a flexible shorthand for setting the range of every coordinate for each plot command, that supersedes the global settings. The notation sets the ranges within square brackets, with the variables listed in a particular order. In rectangular coordinates, the order is x, y, x2, y2. Here's how it works:

```
unset key
set grid lw 2
plot [-5 : 5] [-10 : 30] x**2 lw 4
```



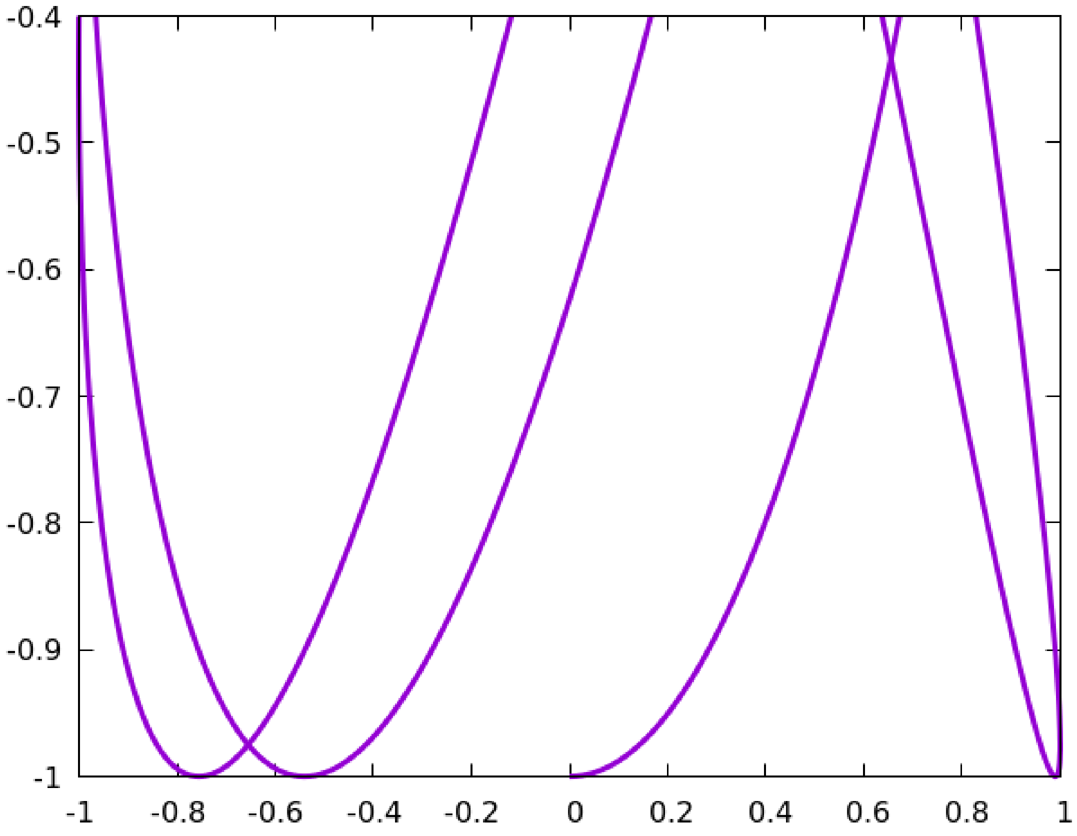
In the last example we used the bracket range notation to set the xrange to go from -5 to 5 and the yrange to cover -10 to 30. The range notation can also be used with parametric plots; in this case the order is t (the parameter), x, y, x2, y2. Let's plot our first parametric example again, but this time limiting the trange. Since we only give one range command below, it applies to t, with x and y left to the defaults:

```
set samples 1000
set parametric
unset key
plot [0 : pi] sin(7*t), cos(11*t) lw 3
```



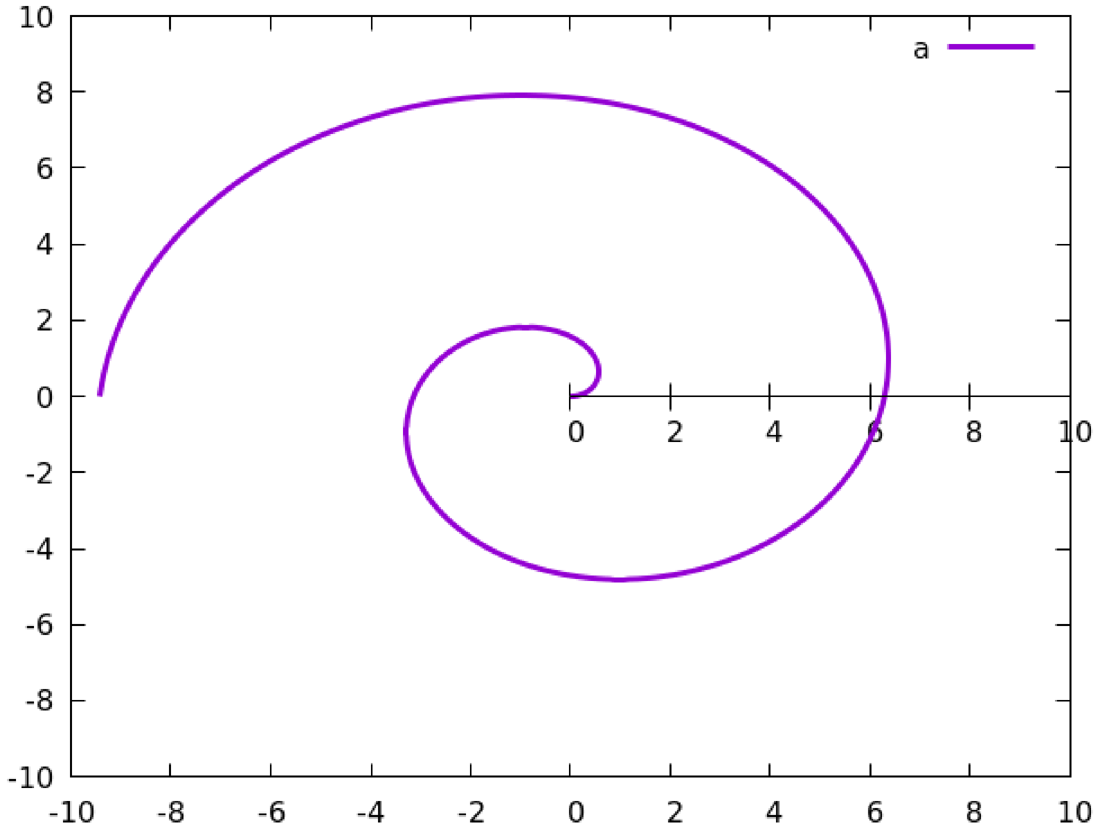
To skip one coordinate in the list of range commands, use an empty bracket. Also, to use the default limit on one side of the range, you can leave it blank. For example, `[5 : ]` would mean that the coordinate in question should start at 5 and end at its normal default value. In the example below we illustrate both of these shorthands. In this parametric plot, we set the parameter t to go from its default, which is -5, to -π; the x-coordinate will take its default values ; and the y-coordinate will range from -1 to -4:

```
set samples 1000
set parametric
unset key
plot [ : -pi] [] [-1 : -.4] sin(7*t), cos(11*t) lw 3
```

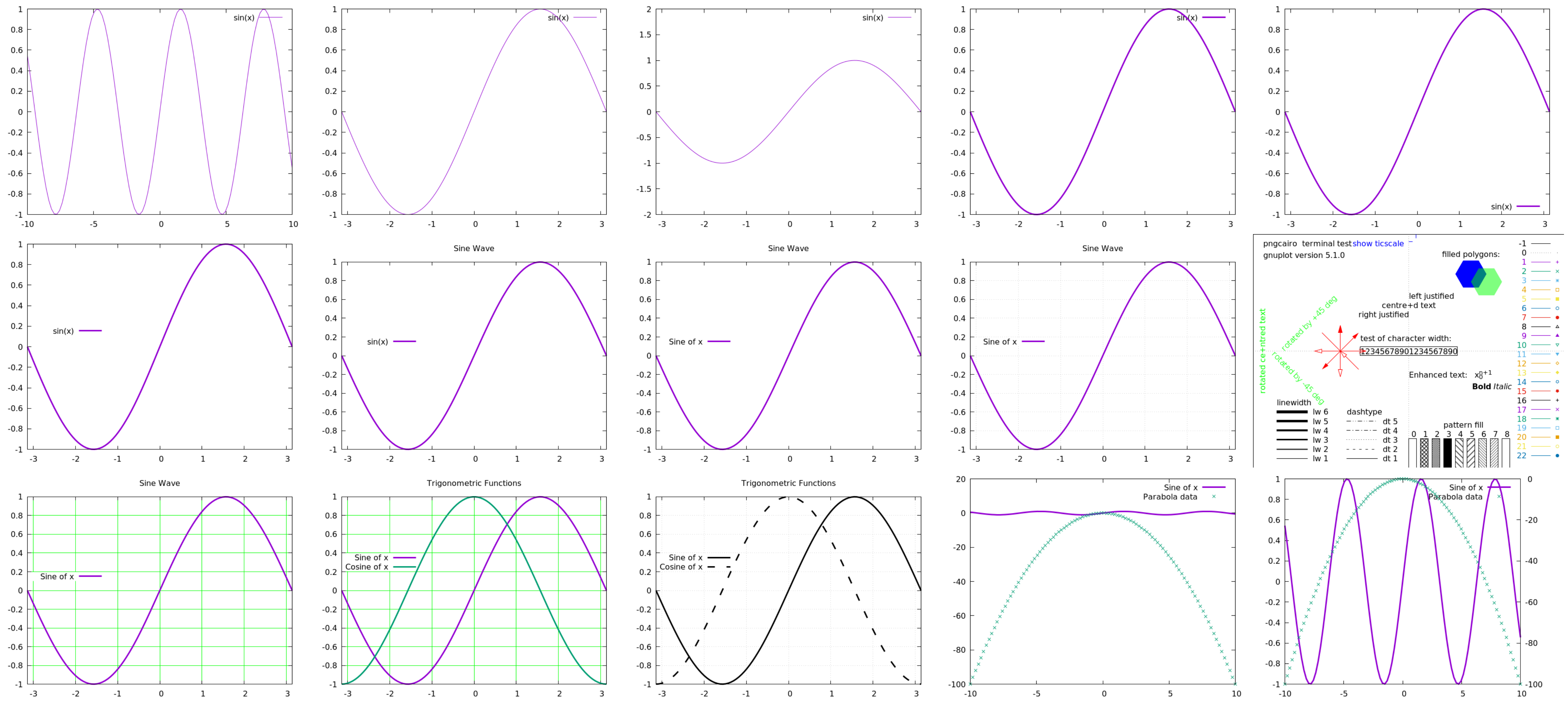


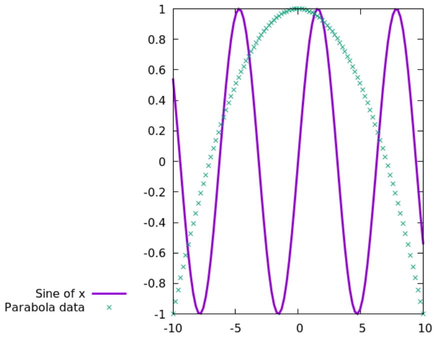
The local range commands, as we are calling them, also allow you to redefine the symbol used for the dummy, or independent variable. (In rectangular coordinates, this variable is x; in both polar and parametric plots, it is called t.) This can be set globally with the `set dummy` command, but can also be set locally, as we show in the current example. We'll replot the Archimedes spiral that we used to introduce polar coordinates, but this time using a smaller angular range, and we'll redefine the angle coordinate to be "a":

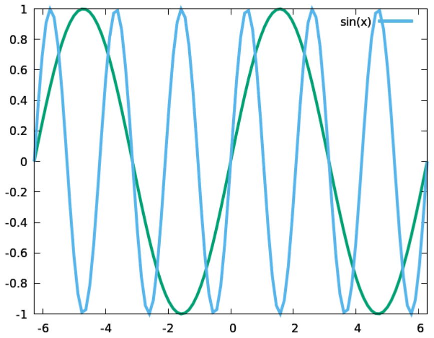
```
set polar
set samples 500
plot [a = 0 : 3*pi]  a lw 3
```

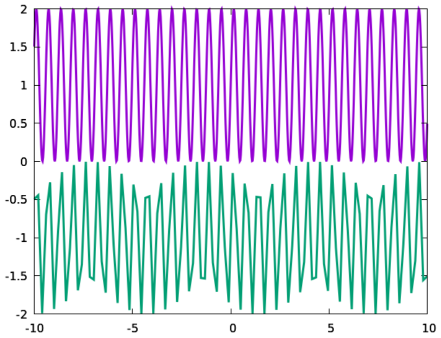


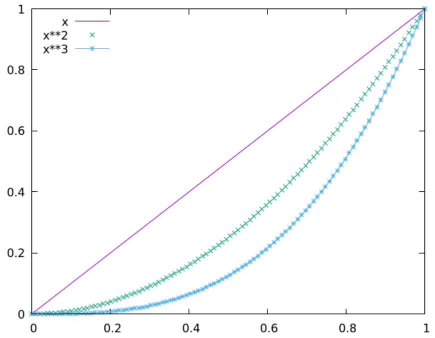
Index of Plots

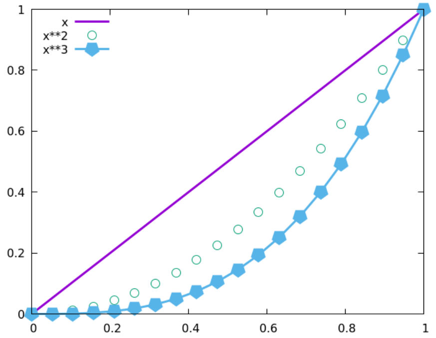


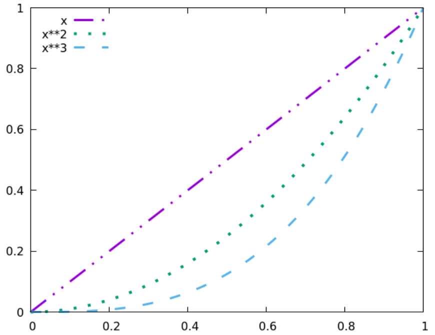

Sine of x
Parabola data

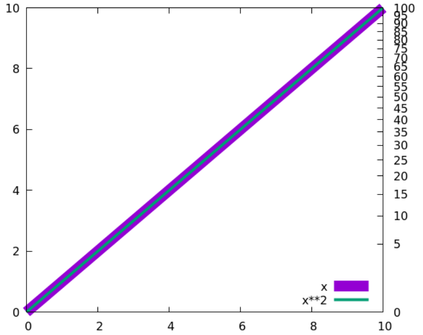

sin(x)

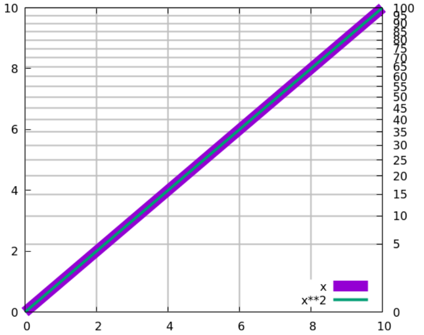


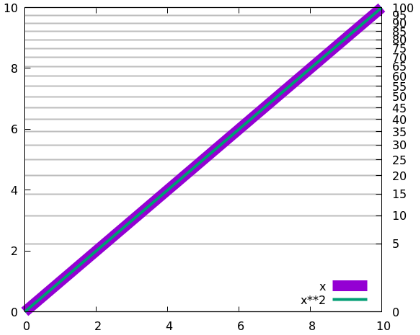

x
x**2
x**3

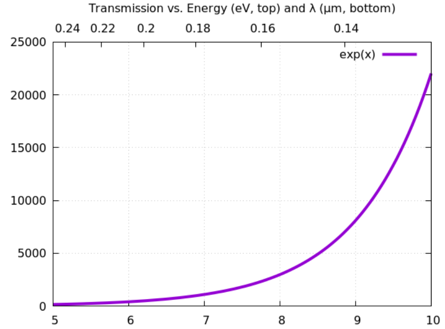

x
x**2
x**3

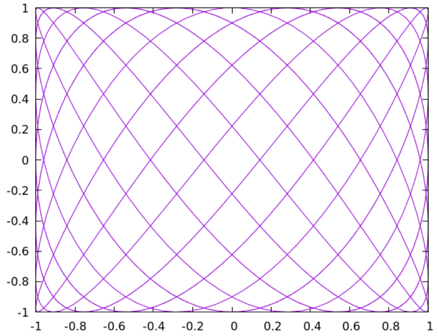

x
x**2
x**3

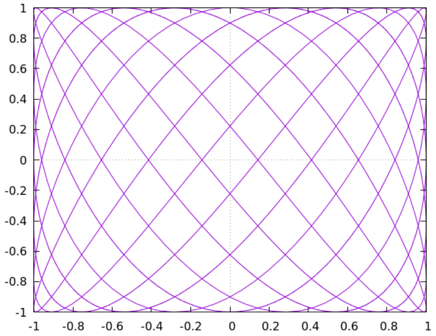

x
x**2

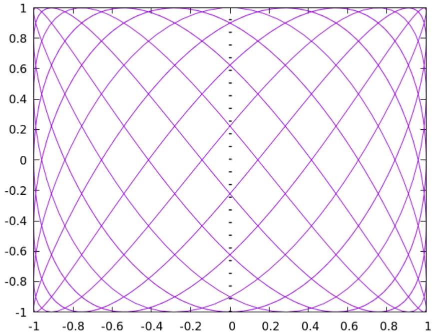

x
x**2

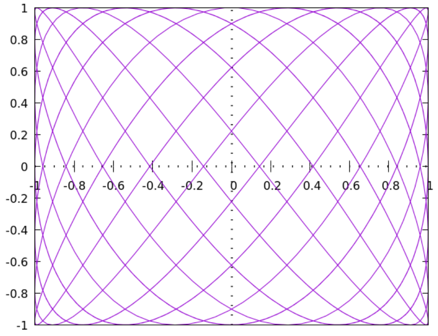

x
x**2

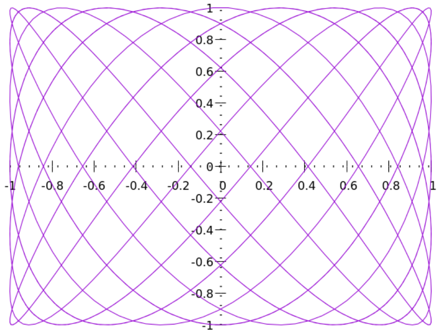

Transmission vs. Energy (eV, top) and λ (μm, bottom)
exp(x)

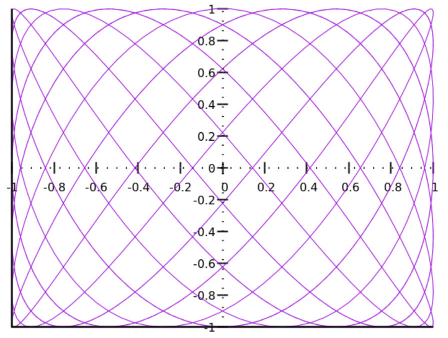


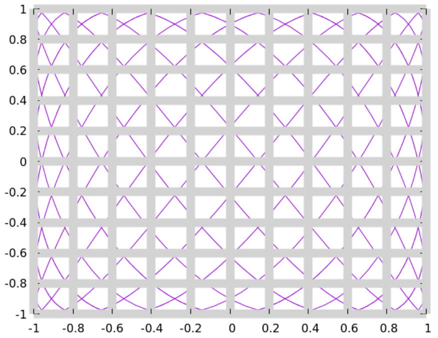


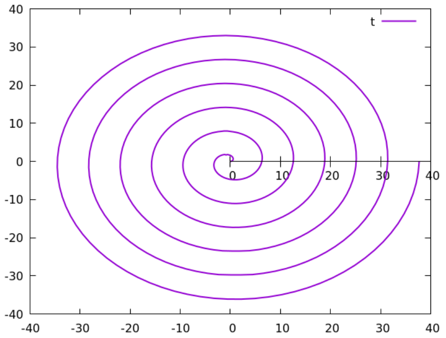


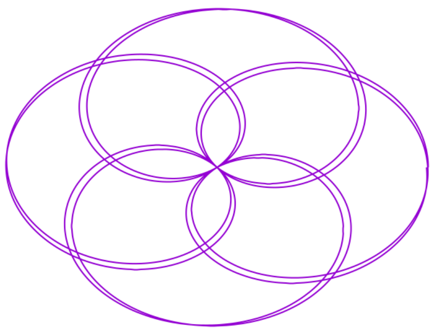


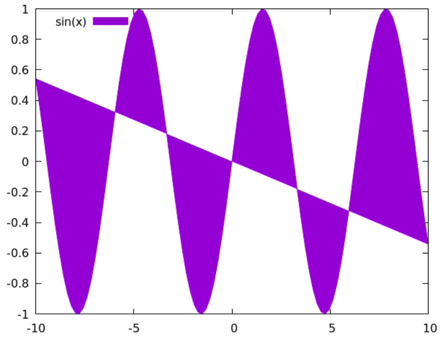


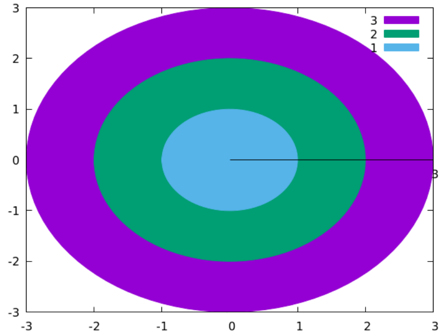


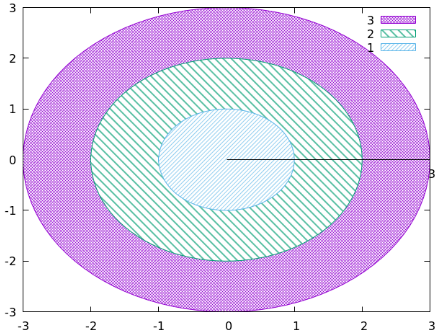


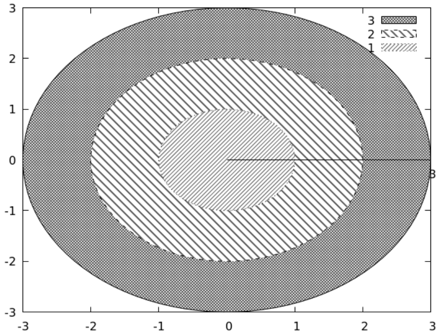

t

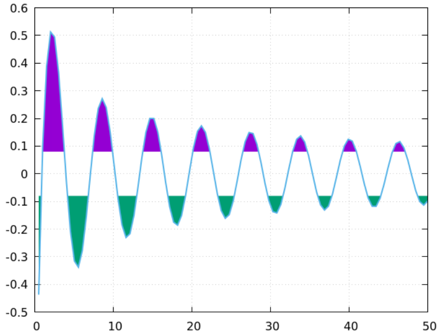


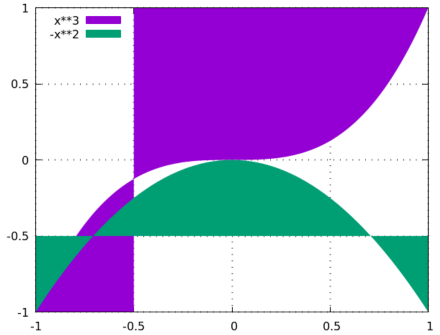

sin(x)

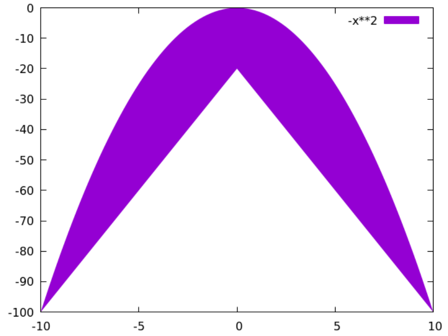

3
2
1

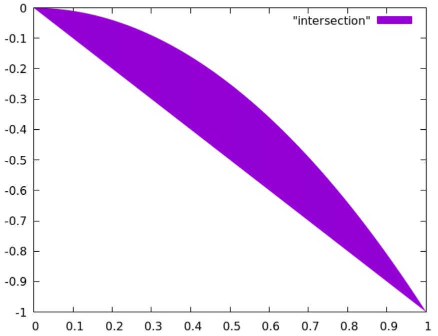

3
2
1

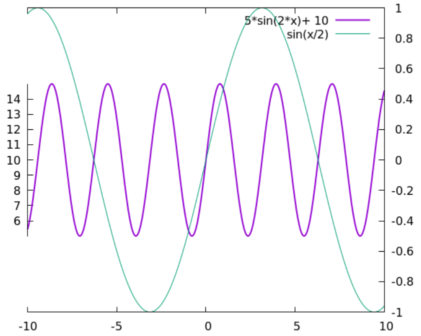

3
2
1

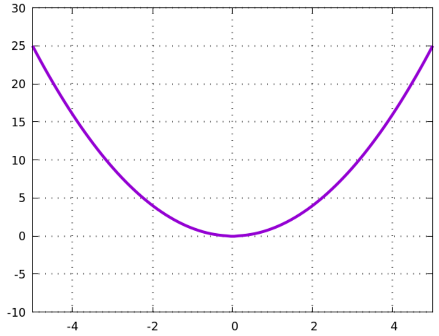


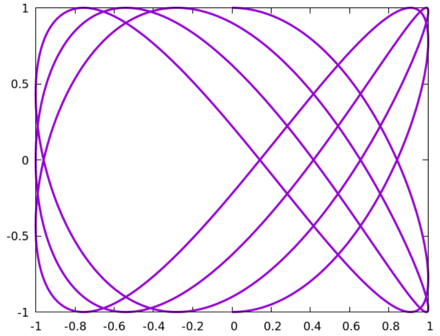

x**3
-x**2

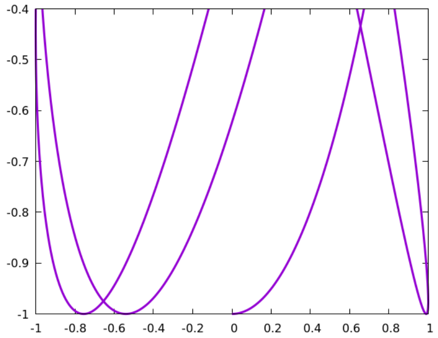

-x**2

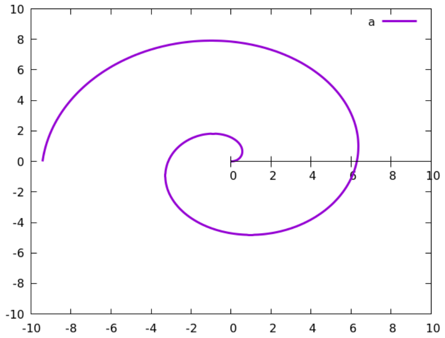

"intersection"


5*sin(2*x)+ 10
sin(x/2)






-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


a
10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-8
-6
-4
-2
0
2
4
6
8
10