Covers gnuplot version 5

SAMPLE CHAPTER

gnuplot IN ACTION

Understanding data with graphs

SECOND EDITION

Philipp K. Janert

MANNING

Gnuplot in Action
by Philipp K. Janert

Chapter 2

brief contents

2 Tutorial: essential gnuplot

This chapter covers

- Invoking gnuplot
- Plotting functions and data
- Saving and exporting
- Managing options
- Getting help

This chapter introduces gnuplot's most important features: generating plots, saving them to a file, and exporting graphs to common graphics file formats. I'll also explain briefly how to set options and how to access gnuplot's built-in documentation. This chapter covers those commands that you'll find yourself using almost every time you start up gnuplot. In the next couple of chapters, you'll learn about further features for graphical analysis and how to manage data sets and files. By the end of chapter 4, you'll know most of the commands you'll use on a day-to-day basis.

Are you surprised that just a few chapters are sufficient to get you this far? Congratulations! You just discovered why gnuplot is cool: it makes easy things easy, and hard things possible. This chapter and the next two cover the easy parts. As to the hard parts … well, that's what the rest of this book is about.

2.1 Simple plots

Because gnuplot is a plotting program, it should come as no surprise that the most important gnuplot command is `plot`. It can be used to plot both functions (such as $\sin(x)$) and data (typically from a file). The `plot` command has a variety of options and subcommands, through which you can control the appearance of the graph as well as the interpretation of the data in the file. The `plot` command can even perform arbitrary transformations on the data as you plot it.

2.1.1 Invoking gnuplot and first plots

Gnuplot is a *text-based* plotting program: you interact with it through command-line-like syntax, as opposed to manipulating graphs using the mouse in a WYSIWYG fashion.[2] Gnuplot is also *interactive*: it provides a prompt at which you type commands. When you enter a complete command, the resulting graph immediately pops up in a separate window. This is in contrast to a graphics *programming language* (such as PIC), where writing the command, generating the graph, and viewing the result are separate activities, requiring separate tools. Gnuplot has a history feature, making it easy to recall, modify, and reissue previous commands. The entire setup encourages you to play with the data: making a simple plot, changing some parameters to hone in on the interesting sections, eventually adding decorations and labels for final presentation, and in the end exporting the finished graph in a standard graphics format.

If gnuplot is installed on your system, it can usually be invoked by issuing the command

```
shell> gnuplot
```

at the shell prompt. (Check appendix A for instructions on obtaining and installing gnuplot, if your system doesn't have it already.) Once launched, gnuplot displays a welcome message and then replaces the shell prompt with a `gnuplot>` prompt. Anything entered at this prompt is interpreted as gnuplot commands until you issue an `exit` or `quit` command, or type an end-of-file (EOF) character, usually by pressing Ctrl-D on Unix.

Probably the simplest plotting command you can issue is

```
plot sin(x)
```

(Here and in the following, the `gnuplot>` prompt is suppressed to save space. Any code shown should be understood as having been entered at the `gnuplot>` prompt, unless otherwise stated.)

On Unix running a GUI (an arbitrary window manager running on top of X11), this command opens a new window showing the resulting graph, something like figure 2.1. Note how gnuplot has selected a "reasonable" range for the x values automatically (by default, from -10 to +10) and adjusted the y range according to the values of the function.

[2] The Windows version of gnuplot contains a menu you can use to build up command strings using the mouse.

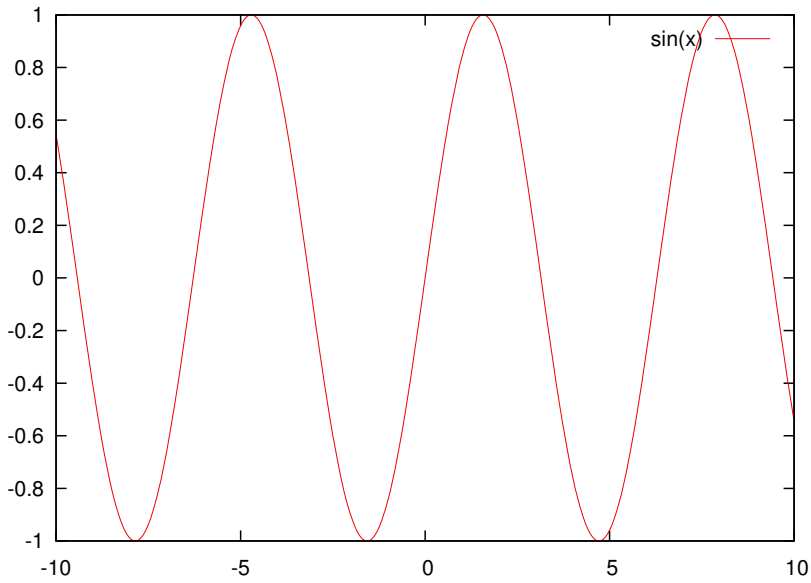


Figure 2.1 Your first plot: `plot sin(x)`

Let's say you want to add more functions to plot together with the sine. You recall the last command (using the up-arrow key or Ctrl-P for "previous") and edit it to give

```
plot sin(x), x, x-(x**3)/6
```

This will plot the sine together with the linear function x and the third-order polynomial $x - \frac{1}{6}x^3$, which are the first few terms in the Taylor expansion of the sine.[3] (Gnuplot's syntax for mathematical expressions is straightforward and similar to that found in almost any other programming language. Note the `**` exponentiation operator, familiar from Fortran or Perl. Section 3.2 contains tables of all available operators and their precedences.) The resulting plot (see figure 2.2) is probably *not* what you expected.

The range of y values is far too large, compared to the previous graph. You can't even see the wiggles of the original function (the sine wave) anymore. Gnuplot adjusts the y range to fit in all function values, but for this plot, you're only interested in points with small y values. So, you recall the last command again (using the up-arrow key) and define the plot range you're interested in:

```
plot [][-2:2] sin(x), x, x-(x**3)/6
```

[3] A Taylor expansion is a local approximation of an arbitrary, possibly complicated, function in terms of powers of x. You won't need this concept in the rest of this book. Check your favorite calculus book if you want to know more.

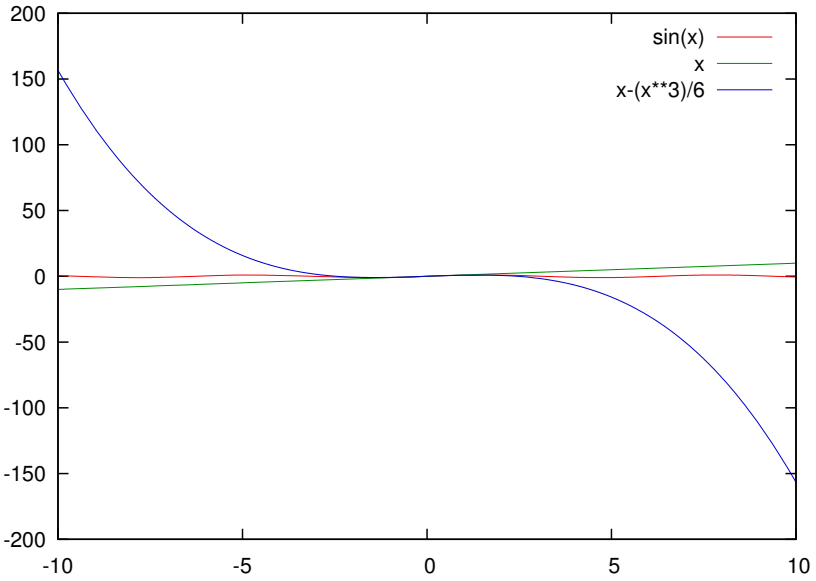


Figure 2.2 An unsuitable default plot range: `plot sin(x), x, x-(x**3)/6`

The range is given in square brackets *immediately after* the `plot` command. The first pair of brackets defines the range of x values (leave it empty, because you're happy with the defaults in this case); the second restricts the range of y values shown. This results in the graph shown in figure 2.3.

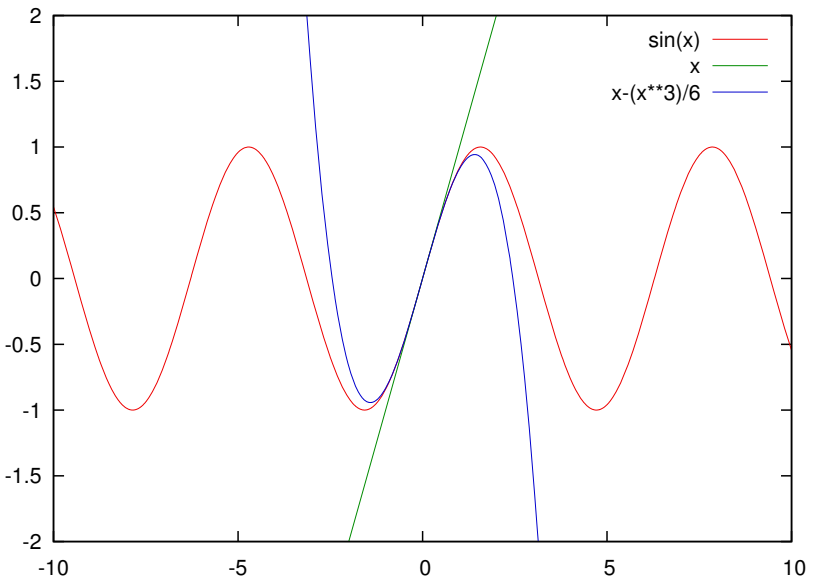


Figure 2.3 Using explicit plot ranges: `plot [][-2:2] sin(x), x, x-(x**3)/6`

You can play much longer with function plots, zoning in on different regions of interest and trying different functions (section F.4 contains a table of all built-in mathematical functions). But let's move on and discuss what gnuplot is most useful for: plotting data from a file.

2.1.2 Plotting data from a file

Gnuplot reads data from text files. The data is expected to be *numerical* and to be stored in the file in *whitespace-separated columns*. Lines beginning with a hash mark (#) are considered to be comment lines and are ignored. The following listing shows a typical data file containing the share prices of two fictitious companies, with the equally fictitious ticker symbols PQR and XYZ, over a number of years.

Listing 2.1 A typical data file: stock prices over time (file: prices)

```
# Average PQR and XYZ stock price (in dollars per share) per calendar year
1975     49    162
1976     52    144
1977     67    140
1978     53    122
1979     67    125
1980     46    117
1981     60    116
1982     50    113
1983     66     96
1984     70    101
1985     91     93
1986    133     92
1987    127     95
1988    136     79
1989    154     78
1990    127     85
1991    147     71
1992    146     54
1993    133     51
1994    144     49
1995    158     43
```

The canonical way to think about this is that the x value is in column 1 and the y value is in column 2. If there are additional y values corresponding to each x value, they're listed in subsequent columns. (You'll see later that there's nothing special about the first column. In fact, any column can be plotted along either the x or the y axis.)

This format, simple as it is, has proven to be extremely useful—so much so that long-time gnuplot users usually generate data this way to begin with. In particular, the ability to keep related data sets in the same file is a big help (so you don't need to keep PQR's stock price in a separate file from XYZ's—although you could if you wanted to).

Although whitespace-separated numerical data is what gnuplot expects natively, gnuplot can parse and interpret significant deviations from this norm, including text

columns (with embedded whitespace if enclosed in double quotes), missing data, and a variety of textual representations for calendar dates, as well as binary data. (See chapter 4 for a more detailed discussion of input file formats, chapter 5 for strings, and chapter 8 for the special case when one of the columns represents date/time information.)

Plotting data from a file is simple. Assuming that the file shown in listing 2.1 is called prices and is in the current working directory (typically, the directory from which gnuplot was started), you can type

```
plot "prices"
```

Because data files typically contain many different data sets, you'll usually want to *select the columns* to be used as x and y values. This is done through the `using` directive to the `plot` command:

```
plot "prices" using 1:2
```

This plots the price of PQR shares as a function of time: the first argument to the `using` directive specifies the column in the input file to be plotted along the horizontal (x) axis, and the second argument specifies the column for the vertical (y) axis. If you want to plot the price of XYZ shares in the same plot, you can do so easily (as shown in figure 2.4):

```
plot "prices" using 1:2, "prices" using 1:3
```

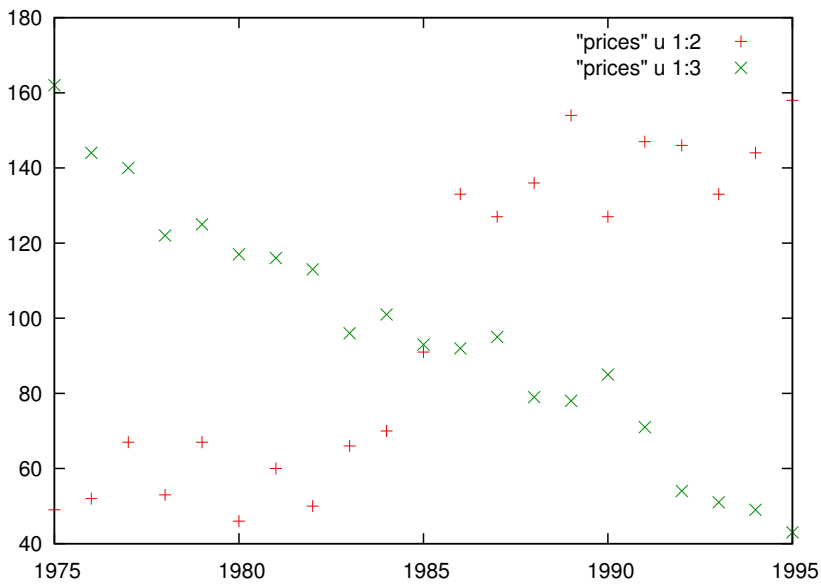


Figure 2.4 Plotting from a file: `plot "prices" using 1:2, "prices" using 1:3`

TIP The `using` directive tells the `plot` command which columns to use. `plot "data" using 1:2` selects the first column for the x axis and the second column for the y axis.

By default, data points from a file are plotted using unconnected symbols. Often this isn't what you want, so you need to tell gnuplot what *style* to use for the data. You do so using the `with` directive. Many different styles are available. Among the most useful are `with linespoints`, which plots each data point as a symbol and also connects subsequent points, and `with lines`, which just plots the connecting lines, omitting the individual symbols:

```
plot "prices" using 1:2 with lines,
➥   "prices" using 1:3 with linespoints
```

TIP The `with` directive to the `plot` command selects the plotting style. The most frequently used styles include `with points`, `with lines`, and `with linespoints`.

This looks good, but it's not clear from the graph which line is which. Gnuplot automatically provides a *key*, which shows a sample of the line or symbol type used for each data set together with a text description, but the default description isn't very meaningful in this case. You can do much better by including a `title` for each data set as part of the `plot` command:

```
plot "prices" using 1:2 title "PQR" with lines,
➥   "prices" using 1:3 title "XYZ" with linespoints
```

This changes the text in the key to the string given as the title (see figure 2.5). The `title` has to come after the `using` directive in the `plot` command. A good way to memorize this order is to remember that you must specify the data set to plot *first* and provide the description *second*: define it first, then describe what you defined.

TIP You can use the `title` directive to place a descriptive string in the graph's legend.

Want to see how PQR's price correlates with XYZ's? No problem; plot one against the other, using PQR's share price for x values and XYZ's for y values, like so:

```
plot "prices" using 2:3 with points
```

You see here that there's nothing special about the first column. Any column can be plotted against either the x or the y axis; you pick whichever combination you need through the `using` directive. Because it makes no sense to connect the data points in the last plot, I chose the style `with points`, which plots a symbol for each data point but no connecting lines (see figure 2.6).

A graph like figure 2.6 is known as a *scatter plot* and can show correlations between two data sets. In this graph, you can see a clear negative correlation: as the stock price of PQR is going up, the price of XYZ is going down. We'll revisit scatter plots and their uses in chapter 13.

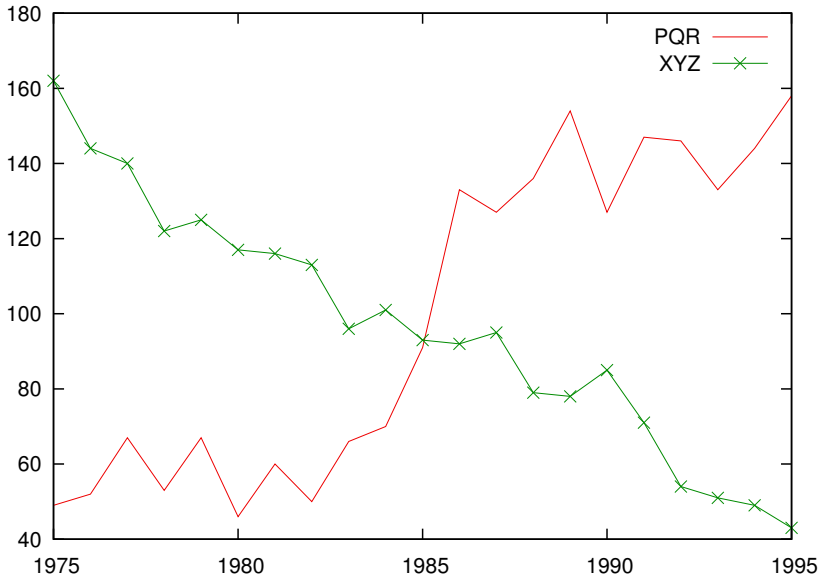


Figure 2.5 Introducing styles and the `title` keyword: `plot "prices" using 1:2 title "PQR" with lines, "prices" using 1:3 title "XYZ" with linespoints`

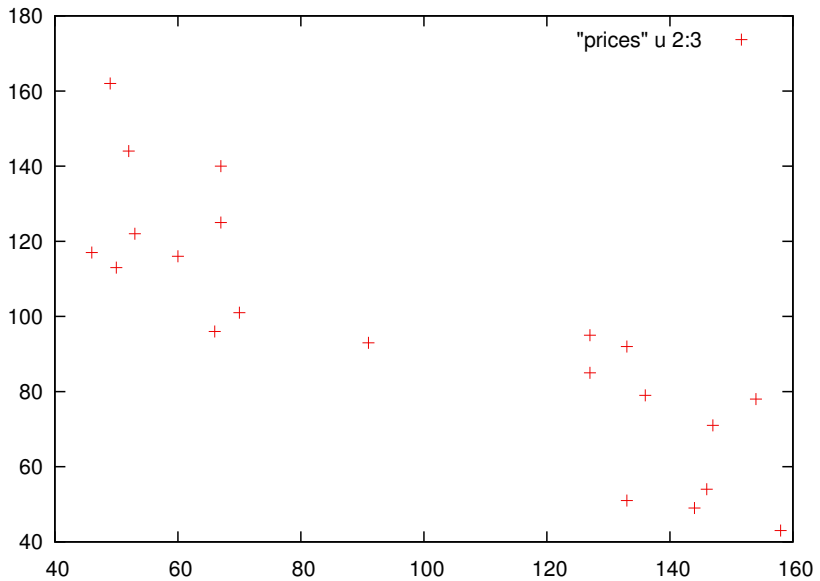


Figure 2.6 Any column can be used for either the x or y axis: `plot "prices" using 2:3 with points`.

Now that you've seen the most important basic commands, let's step back for a moment and quickly introduce some creature comforts that gnuplot provides to the more experienced user.

2.1.3 Abbreviations and defaults

Gnuplot is good at encouraging iterative, exploratory data analysis. Whenever you complete a command, the resulting graph is shown immediately, and all changes take effect at once. Writing commands isn't a different activity from generating graphs, and there's no need for a separate viewer program. (Graphs are also created almost instantaneously; only for data sets including millions of points is there any noticeable delay.) Previous commands can be recalled, modified, and reissued, making it easy to keep playing with the data.

Gnuplot offers two more features to the more proficient user: *abbreviations* and *sensible defaults*. Any command and subcommand or option can be abbreviated to the shortest, non-ambiguous form. So the command

```
plot "prices" using 1:2 with lines,
➥    "prices" using 1:3 with linespoints
```

is more likely to be issued as

```
plot "prices" u 1:2 w l, "prices" u 1:3 w linesp
```

You can shorten the `linespoints` style description even further to `lp`, so the command becomes

```
plot "prices" u 1:2 w l, "prices" u 1:3 w lp
```

This compact style is useful when you're doing interactive work, and you should master it. From here on, I'll increasingly use it. (You can find a table with the most frequently used abbreviations at the beginning of this book in the "About this book" section.)

But this is still not the most compact form possible. Whenever part of a command isn't given explicitly, gnuplot first tries to interpolate the missing values with values the user has provided; failing that, it falls back to sensible defaults. You've already seen how gnuplot defaults the range of x values to `[-10:10]` but adjusts the y range to include all data points.

> **TIP** Abbreviations for common keywords facilitate quick, interactive, iterative work. You can find a table of common abbreviations in the "About this book" section at the start of the book.

Whenever a filename is missing, the most recent filename is interpolated. You can use this to abbreviate the previous command even further:

```
plot "prices" u 1:2 w l, "" u 1:3 w lp
```

Note that the second set of quotation marks *must* be there.

In general, any user input (or part of user input) remains unaffected until explicitly overridden by subsequent input. The way the filename is interpolated in the preceding example is a good example of this behavior. In later chapters, you'll see how options can be built up step by step, by subsequently providing values for different suboptions. Gnuplot helps to keep commands short by remembering previous commands as much as possible.

One last example concerns the `using` directive. If it's missing entirely and the data file contains multiple columns, gnuplot plots the second column versus the first (this is equivalent to `using 1:2`). If a `using` directive is given but lists only a single column, gnuplot uses this column for y values and provides x values as integers starting at zero. This is also what happens when no `using` is given and the data file contains only a single column.

2.2 Saving commands and exporting graphics

There are two ways to save your work in gnuplot: you can *save* the gnuplot commands used to generate a plot, so that you can regenerate the plot at a later time. Or you can *export* the graph to a file in a standard graphics file format, so that you can print it or include it in web pages, documents, or presentations.

> **TIP** *Saving* a graph is the act of storing the gnuplot commands used to create the graph. *Exporting* a graph means creating a version of the graph in a commonly used graphics file format (such as PNG, PDF, or SVG).

2.2.1 Saving and loading commands

If you save to a file the commands you used to generate a plot, you can later load them again and regenerate the plot where you left off. Gnuplot commands can be saved to a file using the `save` command:

```
save "graph.gp"
```

This saves the current values of all options, as well as the most recent `plot` command, to the specified file. This file can later be loaded again using the `load` command:

```
load "graph.gp"
```

The effect of loading a file is the same as issuing all the contained commands (including the `plot` command) at the gnuplot prompt.

> **TIP** Never forget to save your commands using `save` so that you can re-create your graph later using `load`.

An alternative to `load` is the `call` command, which is similar to `load` but takes up to nine additional parameters after the filename to load. The parameters are available in the loaded file in the variables `ARG1` through `ARG9`. The special variable `ARG0` is set to

the filename, and the variable `ARGC` holds the number of parameters supplied to `call`.[4] You can use `call` to write simple scripts for gnuplot.

Command files are plain text files, usually containing exactly one command per line. Several commands can be combined on a single line by separating them with a semicolon (`;`)—this also works at the interactive command line. The hash mark (`#`) is interpreted as a comment character: the rest of the line following a hash mark is ignored. The hash mark isn't interpreted as a comment character when it appears inside quoted strings.

The recommended file extension for gnuplot command files is .gp, but you may also find people using .plt instead. Because command files are plain text files, they can be edited using a regular text editor. It's sometimes useful to author them manually and load them into gnuplot (see section 12.2 for further discussion of that point), and they're also used for batch operations (chapter 11) and configurations (chapter 12).

2.2.2 Exporting graphs

The `save` command I just introduced saves the commands required to generate (or regenerate) the graph into a text file, but it doesn't save the graph itself. How do you get the graph that you see in the plot window into a file? It turns out there are three ways to do it: easy, roundabout, and complicated.

USING THE GUI BUTTON

In recent gnuplot versions, most contemporary terminals (including the `wxt` and `qt` terminals) include a GUI button you can use to export a graph to a file. The button opens a standard file-selection dialog where you select the desired name of the target file and the file format (PNG, PDF, or SVG). Easy.

USING THE CLIPBOARD

A roundabout way to save the graph to a file is to take a screenshot of the graph, using your favorite screenshot utility, and save the result as a GIF or PNG file. After all, the graphics window on the screen is nothing more than a bitmap image, and a screenshot saves it to a file.

Gnuplot facilitates this process by offering a GUI button that copies the current content of the plot window to the window manager's clipboard, from which the graph can be pasted into a paint program (such as the Gimp or Microsoft Paint) or saved directly to a file. Only the canvas is copied, without the tool and title bars, and without any of the decorations put on the window by the window manager. Don't be disturbed if nothing appears to be happening if you click the GUI button: there is no visual feedback, but the window's content is copied to the clipboard, from which it can be pasted into other applications. (The details of how to do that depend on your choice of window manager and the way the clipboard is configured on your local computer.) This method may appear a bit unsophisticated, but it has two significant advantages: the

[4] Earlier versions of gnuplot used the special tokens `$0` through `$9`. That syntax is now deprecated.

workflow is simple, and the resulting graph corresponds *exactly* to the graph that was on the screen.

USING TERMINALS

Both the GUI button and the clipboard are relatively new methods for exporting a graph. They're easy to use but don't allow for much flexibility in the appearance of the generated graph. To get full control over the export process, you need to become familiar with gnuplot's *terminal* facility.

In gnuplot parlance, a terminal is a graphics-capable output device. Traditionally, this may have been a specific piece of hardware (such as a pen-and-ink plotter), but today a gnuplot terminal is merely a reference to the underlying graphics library.

> **TIP** A terminal is gnuplot's abstraction for a graphics-capable end point. Terminals can be interactive (screen oriented) or file-based. All graphics output must be directed to a terminal.

Contemporary terminals come in two flavors: interactive and file-based. Interactive terminals create the graphs that you see on the screen. On Linux, you have a choice among three interactive terminals, each built using a different widget set: the `qt` terminal uses Qt, the `wxt` terminal uses wxWidgets, and the old `x11` terminal is a pure Xlib application. (Both the `qt` and the `wxt` terminals also exist on Windows and Mac OS X, together with platform-specific terminals.)

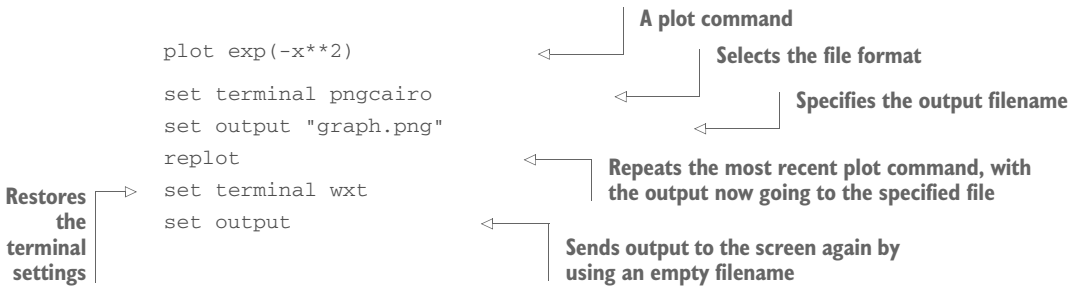
But to export a graph to a file, you need to employ a file-based terminal that can generate output in the desired output format (PNG, PDF, SVG, and so on). Most file-based terminals accept a large number of options through which you can control various aspects (such as the size) of the resulting graph. These options are covered in detail in chapter 10.

All of this is straightforward. But there is one stumbling block: exporting a graph via a file-based terminal *requires multiple steps*. In order to export a graph with a gnuplot terminal, *several commands* must be executed in proper sequence. This may come as a surprise, because "exporting to a file" is a single, atomic operation in most other applications these days. Gnuplot is different.

The full sequence of steps is shown in listing 2.2. Let's step through it:

1. Begin with an arbitrary `plot` command.
2. Select the desired file-based terminal, using the `set terminal` command.
3. Specify the name of the output file, using the `set output` command.
4. Regenerate the last plot, this time sending it to the file-based terminal and the named file.
5. Restore the interactive terminal again, using both `set terminal` and the `set output` command.

Listing 2.2 Complete workflow to export to file



This is the first time you've encountered gnuplot's `set` command—I'll talk about it in more detail in the next section (and subsequently in almost every chapter of this book!). For now, it's enough to understand that it sets a parameter (such as `terminal`) to a value. But—and this is often forgotten—it does *not* generate a plot! The only commands that do so are `plot`, `splot` (which is used for three-dimensional graphs, discussed in appendix C), and `replot` and `refresh` (both of which repeat the most recent `plot` or `splot` command).

I'd like to emphasize three things about listing 2.2:

- The file format and the output filename must be specified separately, using `set terminal` and `set output`, respectively.
- The graph must be sent to the newly created file in a separate command, using `plot`, `replot`, or an equivalent command.
- Afterward, the interactive terminal must be restored explicitly. (This is easy to forget.)

As long as you keep these three items in mind, creating files via terminals won't pose any difficulties. (In chapter 11, you'll see how to use scripts and macros to facilitate this process.)

TIP Be sure to complete *all* the required steps, and in the proper order. Typical mistakes are forgetting to issue a `replot` command (resulting in an empty file) and forgetting to restore the original terminal at the end.

Before leaving this section, one last word of advice: always save the commands used to generate a plot to a command file *before* exporting to a printable format. Always. It's almost guaranteed that you'll want to regenerate the plot to make a minor modification later (such as fixing a typo in a label, adding one more data set, or adjusting the plot range slightly). This can only be done from the commands saved to file using `save`, not from plots exported to a graphics file. We'll come back to this topic several times.

TIP Don't just store the finished graph. Always save the commands required to generate it as well, so you can come back and modify it later.

2.3 Managing options with set and show

Gnuplot has relatively few commands (such as the `plot`, `save`, and `load` commands you've already encountered) but a large number of *options*. These options are used to control everything from the format of the decimal point to the name of the output file. More than 100 such options are available, along with countless sub-options for each. Check the entries for `set` and `show` in the gnuplot standard reference manual for the complete list.

TIP Many aspects of gnuplot's behavior can be controlled through options. Use the `set` command to change their value; use `show` to see their current value.

The three commands used to manipulate individual options are as follows:

- `show`—Displays the current value of an option
- `set`—Changes the value of an option
- `unset`—Disables a specific option or returns it to its default value

There's also a fourth command, `reset`, which returns *all* options to their default values. The only options not affected by `reset` are those directly influencing output generation: `terminal` and `output`.

The syntax of all three commands is straightforward. To assign a new value to an option, use the `set` keyword followed by the name of the option and the new value. The `show` command takes only a single argument: the name of the option you want to inspect.

The `show` command is also used more generally to display all kinds of information about gnuplot's internal state, not just options that can be changed using `set`. For example, `show variables` displays all variables that are defined in the current session together with their values, and `show functions` lists all user-defined functions.

Another useful command is

```
show version long
```

This prints the current version of gnuplot, together with a copyright notice and pointers to the online documentation. More important, it also shows the compile-time flags that gnuplot was compiled with. This is particularly relevant because some features have only recently been added to gnuplot and may not be enabled on all platforms. You can use `show version long` to see which flags your version of gnuplot was built with.

Finally, you can use `show all` to see a listing of all possible options and their values. Be prepared for a long listing!

TIP You can use the `show` command to learn about your gnuplot installation and to inspect the current state of a gnuplot session.

2.4 Getting help

Gnuplot has an extensive built-in, online help system (online in the sense that it's accessible from within the gnuplot session; it has nothing to do with network connectivity to the internet). To get started, enter `help` at the gnuplot prompt. Alternatively, you can go directly to the reference page for a specific command or option by entering the name of the command or option as an argument to the `help` command. For example, to learn about the `plot` command, you'd use

```
help plot
```

The online help is detailed and comprehensive, so you should become familiar with it. But keep in mind that it's a *reference*, not a tutorial. If you know the name of the command or option you're looking for, it's great. But if you want to find all relevant options for a specific task, navigating the online help can be frustrating.

TIP Type `help`, followed by a command or option name, to access gnuplot's built-in reference documentation.

2.5 Summary

In this chapter, you learned how to do the most important things with gnuplot: plotting, saving, and exporting. You also learned how to set and inspect options and how to access the built-in reference documentation.

This means you can already do the three most important things for day-to-day work: generate a plot, save it to file, and export it. In the next chapter, you'll begin to learn how to use gnuplot to analyze data and understand what it's telling you.

OPEN SOURCE/DATA VISUALIZATION

gnuplot IN ACTION Second Edition

Philipp K. Janert

Free eBook
SEE INSERT

Gnuplot is an open-source graphics program that helps you analyze, interpret, and present numerical data. Available for Unix, Mac, and Windows, it is well-maintained, mature, and totally free.

gnuplot in Action, Second Edition is a major revision of this authoritative guide for developers, engineers, and scientists. The book starts with a tutorial introduction, followed by a systematic overview of gnuplot's core features and full coverage of gnuplot's advanced capabilities. Experienced readers will appreciate the discussion of gnuplot 5's features, including new plot types, improved text and color handling, and support for interactive, web-based display formats. The book concludes with chapters on graphical effects and general techniques for understanding data with graphs. It includes four pages of color illustrations. 3D graphics, false-color plots, heatmaps, and multivariate visualizations are covered in chapter-length appendixes available in the eBook.

What's Inside

- Creating different types of graphs in detail
- Animations, scripting, batch operations
- Extensive discussion of terminals
- Updated to cover gnuplot version 5

No prior experience with gnuplot is required. This book concentrates on practical applications of gnuplot relevant to users of all levels.

Philipp K. Janert, PhD, is a programmer and scientist. He is the author of several books on data analysis and applied math and has been a gnuplot power user and developer for over 20 years.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/books/gnuplot-in-action-second-edition

"The highly anticipated, updated version of my go-to-for-everything book on gnuplot."
—Ryan Balfanz, Shift Medical, Inc.

"The essential guide for newcomers and the definitive handbook for advanced users."
—Zoltán Vörös
University of Innsbruck

"Learn how to use gnuplot to convert meaningful data into attention-grabbing visualizations that communicate your message quickly and accurately."
—David Kerns
Rincon Research Corporation

"An accessible guide to gnuplot and best practices of everyday data visualization."
—Wesley R. Elsberry, PhD
RealPage, Inc.

ISBN 13: 978-1-63343-018-1
ISBN 10: 1-63343-018-9
54499
9 781633 430181

MANNING $44.99 / Can $51.99 [INCLUDING eBOOK]