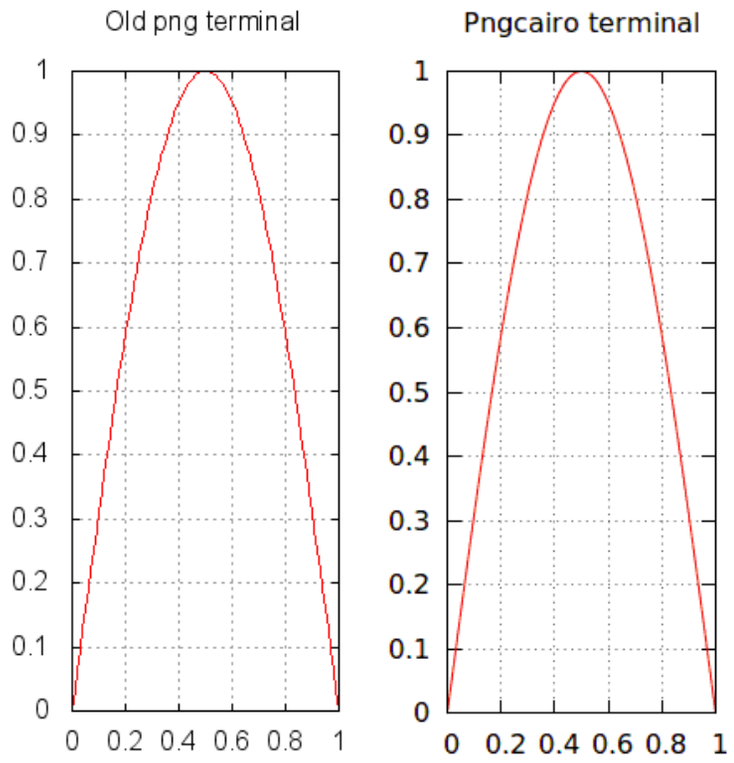
Chapter 9: Gnuplot and LaTeX

TEX has been the tool of choice for the creation of papers and documents for mathematicians, physicists, and other authors of technical material for many years. Presumably, this includes many of the readers of this book. Although it takes some effort and study to learn how to use this venerable work of free software, its devotees become addicted to its ability to produce publication-quality manuscripts in a plain text, version-control friendly format.

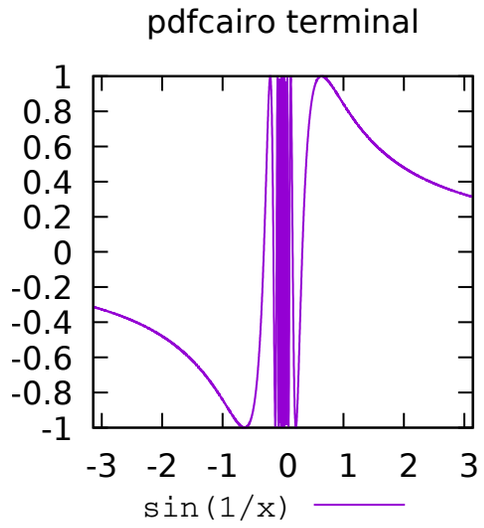
Most TEX users use LATEX, which is a set of commands and macros built on top of TeX that allow automated cross-referencing, indexing, creation of tables of contents, and automatic formatting of many types of documents. TEX, LATEX, a host of associated utilities, fonts, and related programs are assembled into a large package called TEX Live. It's available through the package managers of many Linux distributions, but to get an up-to-date version, one often needs to download it from its maintainers directly.

One reason gnuplot is so popular with scientists, mathematicians, and engineers is its ability to interoperate so intimately with LATEX. Just about any plotting program can make image files that LATEX can include in documents. But gnuplot can work with LATEX in a much deeper way that helps authors and publishers create technical documents with a harmonious blending of text, math, and graphs.



If you're doing things the simple way, by using gnuplot to create a file that you then include using LaTeX's `\includegraphics` command (for example), you should ensure that you're using one of gnuplot's high-quality output formats. This will lead to a document with higher resolution or better appearing figures, more suitable for publication. Generally speaking, these are the terminals that are based on the Cairo libraries: `pdfcairo`, `epscairo`, and `pngcairo`; the latter is the one used for most of the example illustrations in this book. Newer versions of gnuplot have generally replaced the older, plain `png` (etc.) terminals with these, so this is not usually an issue, but you should be aware of the quality difference; the figure shows a plot using the old `png` terminal compared with the same plot using the `pngcairo` terminal. In the Cairo terminals, text and anti-aliasing is handled better, and the drawing of curves in plots is vastly improved. The curve in the older version is much more jagged, compared with the smooth rendering from the Cairo library. `pngcairo` also supports transparency and Unicode, so it is more flexible and takes good advantage of gnuplot's newer features. To see the list of terminals supported by your compilation of gnuplot, just type `set term`. To select, for example, the Cairo-based PNG format, begin your script with `set term pngcairo`.

Some Other Terminals



The `pdfcairo` terminal offers all the features of the `pngcairo` terminal (transparency, Unicode), with the additional benefit of resolution-independence. The PDF-based figures that this terminal produces can be scaled up arbitrarily without loss of quality. They consist of drawing and character-placement instructions rather than the list of colored dots comprising a bitmapped format such as PNG. You can zoom in to this document as far as your PDF reader will allow, and observe that the figure made using the `pdfcairo` terminal remains completely smooth, just as the text that you are reading. Compare with the `pngcairo` figure above, which becomes a bit fuzzy as you zoom in. For this reason, some journals prefer that you supply figures in PDF or another resolution-independent format, which gnuplot makes easy to do. One final note: the Postscript standard supports neither transparency nor simple use of Unicode. Therefore, it is generally simpler to use the `pdfcairo` terminal rather than setting up a workflow where you are creating Postscript files and converting them to PDF.

One drawback to figures in PDF format is that they aren't supported on the Web (entire PDF documents are supported well, but HTML with PDF figures will not work smoothly). Fortunately, gnuplot supports an SVG terminal (`set term svg`), which supports transparency and Unicode, and is as easy to use as any of the other terminals. SVG is also resolution-independent, and, as SVG support is finally widespread among browsers, it's a good choice for putting graphs up on the web. Gnuplot's SVG files can also feature interactivity, as we saw in a previous example implementing "hypertext" labels.

If you have no interest in LaTeX you can safely skip the rest of this chapter, because there's nothing else here crucial for

gnuplot knowledge. If you happen to be evaluating document preparation systems, however, you might want to look this chapter over, because the combination of LaTeX with gnuplot provides some unique capabilities.

In order to reproduce the examples before this chapter, all you needed to have installed was gnuplot. To try out the techniques in this chapter, of course, you'll need to have TeX installed, as well. This is a very large installation — much larger than gnuplot. The modern way to get all the TeX bits and pieces is to install a bundle called TeX Live. If you're on Linux, you can probably find a new enough version of TeX Live through your package manager. If you want a really up-to-date version, or need to run it on another operating system, you may want to download it from its maintainers directly.

As this is not a book about TeX, we won't spend much time explaining how to use the language and the various TeX engines; you will be assumed to have it installed and to possess a basic knowledge of TeX markup and of how to process documents. But we do provide complete gnuplot scripts and LaTeX documents for each example. This chapter is laid out in a somewhat different format from previous ones. Our structure up to now has been to place gnuplot scripts next to the graphs that they produce; but our work in this chapter will involve combining gnuplot with LaTeX in a way that is better served by a less structured arrangement.

Simple Graphics Inclusion

Since this book is itself a PDF document that uses LaTeX, as well as other tools, in its processing, this chapter will be somewhat recursive in nature. The example above, of the inclusion of a PDF figure in a LaTeX document, uses the `wrapfig` LaTeX package,

which allows the text to wrap around the image, which can be floated to the right or the left. Here is a simplified document that produces the example. This book is processed with `lualatex`, but other engines should work just as well, for example `pdflatex` or `xelatex`.

```
\documentclass[12pt]{article}
\usepackage{graphicx}
\usepackage{wrapfig}
\usepackage{blindtext}

\begin{document}

\begin{wrapfigure}{r}{0.25\textwidth}
\includegraphics[width=0.25\textwidth]{pdfcairo}
\end{wrapfigure}

\blindtext

\end{document}
```

Open file

The `blindtext` package is good for testing and generating examples: it spits out filler text. Run this document through LaTeX and see what you get! It's included as an attachment, as the gnuplot scripts in previous chapters, which should make opening and saving the file more convenient. You need, of course, to have an image in your directory called "pdfcairo.pdf" for this to work unaltered. Actually, as you may know, you can have any type of image that your TeX engine supports, so the highlighted `includegraphics` line will work if you happen to have a PNG image called "pdfcairo.png" in your directory — but, up above, we were interested in demonstrating the inclusion of PDF images.

Of course, this example document doesn't care whether the graphics file comes from gnuplot or anything else. This is the simple technique that you'll use, even if you're making graphs with gnuplot, if you post-process the files with the Gimp or another image processing program.

We turn now to gnuplot terminals that are specifically designed to work with LaTeX.

The tikz Terminal

Before trying this out, you need to make sure that your version of gnuplot has the `tikz` terminal baked in. Since this terminal has some slightly uncommon dependencies, not all versions do. If you compiled gnuplot yourself and don't have

certain libraries installed on your system, the `tikz` terminal will be missing. Type `set term` to see the list of terminals. If `tikz` is not on the list, you can get essentially the same results, with some extra inconvenience, by using the `epslatex` terminal, covered in the next section. However, `tikz` is the modern way, and might be worth the trouble of installing the required libraries and recompiling.

The purpose of the `tikz` terminal, and similar solutions, is to make the fonts used in your graph, in the tic labels, title, and elsewhere, to match the fonts used in the rest of your document. This produces a unified, sophisticated appearance, and makes your text easier to read. It is quite difficult to do this entirely within gnuplot, or by post-processing with an image editor. Even if you could get these programs to find and use the fonts that will be incorporated into your document by LaTeX, there would be no way to match the resulting typographic detail in the main text: the kerning, line breaking, ligatures, and so on. And if you include equation labels in your graph, you would not have access to TeX's mathematical syntax nor any way to produce results comparable to its math output. Your best alternative would be to use the "enhanced text" markup we covered earler, which is cumbersome, unpredictable, and usually leads to results that, compared with TeX's mathematical output, are notably unattractive (but can be convenient in simple cases).

The first step is to tell gnuplot to `set terminal tikz`. Then, you need to direct the output to a file, and the name of the file should end with ".tex" (not an actual requirement, but this will be more convenient). For example, `set output "r3.tex"`. After those two lines, you can create any type of plot you need. The result will be, not a graphics file, such as a PDF or PNG image, but a TeX file, consisting of a set of instructions for LaTeX. You can even read and edit this file in a text editor, to further customize it, for example. Here is an example gnuplot script that uses the `tikz` terminal, and includes some mathematical labels; we create these using TeX syntax:

```
set terminal tikz
set out 'r3.tex'
unset key
set label\
 '\Large$\displaystyle \lim_{x\Longrightarrow0}\frac{\sin(x)}{x}=1$'\
 at graph .55, .75
set title\
  '\Large Illustrating L'Hô\-pi\-tal's Rule: $\frac{\sin(x)}{x}$'
plot [0:15] sin(x)/x lw 2
```

Open file

If you run this through gnuplot and look at the resulting "r3.tex" file, you will see a long list of strange-looking commands. These are drawing commands that will be interpreted by the `tikz` package (actually, the customized `gnuplot-lua-tikz` package included with gnuplot) to produce a resolution-independent, vector plot embedded into your LaTeX document.

A few notes are in order: there is a difference, in gnuplot, between strings inside of double quotation marks and those

inside of single quotation marks. We've usually used double quotes in our examples, but, in this case, we've had to use single quotes. This is because of the backslashes in the TeX commands within the labels: within a double-quoted string in gnuplot, backslashes are not literal, but make the following character special. The other option is to use double quotes and also double the backslashes.

The other detail involves the use of `\displaystyle` in the equation. The displayed math environment is not available within `tikz`, requiring this modifier. Otherwise, you can put pretty much any TeX content you want inside titles and labels; it will all be passed on to LaTeX.

Here is a small document that shows how to include the file:

```
\documentclass[12pt]{article}
\usepackage{gnuplot-lua-tikz}
\usepackage{fontspec}
\usepackage{wrapfig}
\begin{document}
\openup.5em

\begin{wrapfigure}{r}{0.5\textwidth}
\resizebox{3.5in}{!}{\input{r3}}
```

```
\end{wrapfigure}

\noindent The figure to the right provides an illustration of
L'Hô\-pi\-tal's Rule.  Recall that this rule can be applied when
taking the limit as $x\longrightarrow x_a$ of a ratio of two
functions where the ratio approaches the indeterminate form
$\frac{0}{0}$; in the case where both functions are
differentiable at $x_a$, the ratio approaches the ratio of their
derivatives. In the case illustrated both $\sin(x)$ and $x
\longrightarrow0$ as we approach the origin, but the ratio of their
derivatives, $\frac{\cos(x)}{1} \longrightarrow 1$. L'Hô\-pi\-tal's
Rule also applies in the case of the indeterminate form $\frac{\infty}{\infty}$.

\end{document}
```

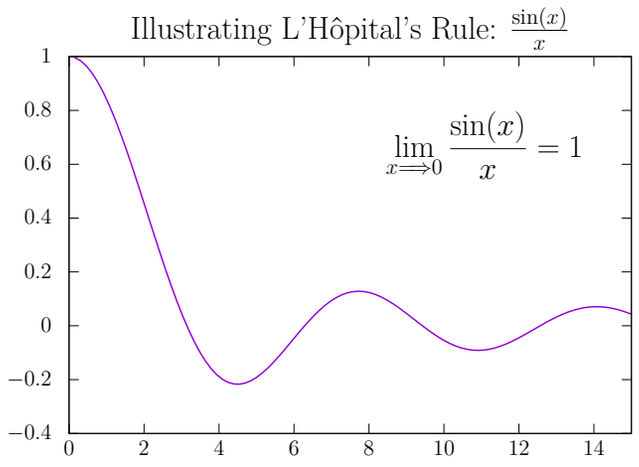
Open file

We've chosen to place the figure inside a `wrapfigure` environment, as before. But since it's not a graphics file, you don't

include it with an `\includegraphics` command; it's a text file full of TEX and `tikz` commands, so you `\input` it. You can, of course, make it any size you like. Note that since the labels in the plot as well as the text contain Unicode characters, you need to include the `fontspec` package (or something equivalent) and process the document using a Unicode-aware engine, such as `lualatex` or `xelatex`.

When you do so, you'll get a PDF file that looks like this:

The figure to the right provides an illustration of L'Hôpital's Rule. Recall that this rule can be applied when taking the limit as $x \longrightarrow x_a$ of a ratio of two functions where the ratio approaches the indeterminate form $\frac{0}{0}$; in the case where both functions are differentiable at x_a, the ratio approaches the ratio of their derivatives. In the case illustrated both $\sin(x)$ and $x \longrightarrow 0$ as we approach the origin, but the ratio of their derivatives, $\frac{\cos(x)}{1} \longrightarrow 1$. L'Hôpital's Rule also applies in the case of the indeterminate form $\frac{\infty}{\infty}$.



If you zoom in, either on the figure reproduced here or in the PDF that you generate by processing the document, you will see that the figure is resolution-independent. Notice also that the fonts used on the graph match the fonts in the text,

and that the math on the graph is genuine LaTeX math. This is because all the graph text is passed to LaTeX for processing; none of it is set by gnuplot.

If you change the font specifications in the document, for example by changing the optional argument in the first line that sets the overall font size, and run it through TeX again, you will see that not only does the font size of the text change, but the text in the graph, including the tic labels, title, and label, change to match. Changing the typefaces has the same effect: the text and graph will always look as if they belong together.

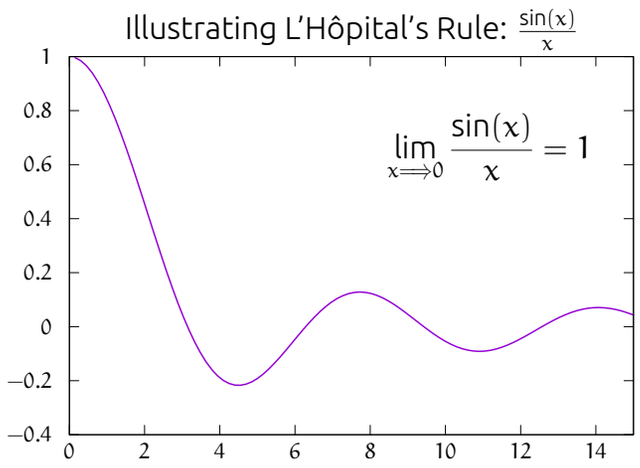
Here's an example. Let's change the preamble of the LaTeX document to use a different typeface as the main font, and, for good measure, load the `euler` package, which uses a different set of fonts for math from the familiar, default Computer Modern:

```
\documentclass[12pt]{article}
\usepackage{gnuplot-lua-tikz}
\usepackage{euler}
\usepackage{fontspec}
\usepackage{wrapfig}
\setmainfont{Ubuntu Light}
\begin{document}
[etc.]
```

Open file

Running this through TeX produces this PDF:

The figure to the right provides an illustration of L'Hôpital's Rule. Recall that this rule can be applied when taking the limit as $x \longrightarrow x_a$ of a ratio of two functions where the ratio approaches the indeterminate form $\frac{0}{0}$; in the case where both functions are differentiable at x_a, the ratio approaches the ratio of their derivatives. In the case illustrated both $\sin(x)$ and $x \longrightarrow 0$ as we approach the origin, but the ratio of their derivatives, $\frac{\cos(x)}{1} \longrightarrow 1$. L'Hôpital's Rule also applies in the case of the indeterminate form $\frac{\infty}{\infty}$.



Notice how the fonts, including those used for math symbols, in the main text still match those used in the graph.

The epslatex Terminal

If your version of gnuplot supports the `tikz` terminal there is probably no need for you to read this section. The `epslatex` terminal is an older way to achieve the same results as we did in the previous section, but with slightly less convenience.

To use the `epslatex` terminal, your gnuplot script should start with `set term epslatex`. As an example, replace the first line of the tikz terminal script with that command. After running the script, gnuplot will create two output files rather than the single file that we usually get. We will have our graph in the form of an encapsulated PostScript (EPS) file with an "eps" extension. We will also have a LaTeX file with the name that we specify in the `set out` command. We should pick a filename with the extension ".tex", as before, to make subsequent processing more convenient. The EPS file will have the same base name (name aside from the extension) as the one given in the set out command.

Assuming our final desired result is a PDF document, we must first transform the encapsulated PostScript file into a PDF file. It doesn't matter what tool we use to accomplish this, but one convenient method available on Linux is to use the command-line tool `epstopdf`, which does just what it says. Another possibility is the `convert` utility that comes with the incredibly useful ImageMagick package. On the Macintosh, if `epstopdf` is not installed, we need merely open the EPS figure in Preview and then save it. Preview converts the file to PDF for display.

The LaTeX file generated by the gnuplot script normally need not be edited nor looked at. Its job is to overlay all the text, including the tic and axis labels, title, etc., onto the plot in the EPS file. If you look at the graph file itself, you will see a bare plot with neither labels nor text.

The LaTeX document will look exactly as it did when using the `tikz` terminal; the command to include the graphics file is

now within the `\input`ted `.tex` file.

Calling gnuplot from LaTeX

This recipe is, in a way, the reverse of the previous examples in this chapter. We are going to learn how to generate gnuplot commands from within a LaTeX document.

The method introduced in this section can be very useful when we want to enfold graphical elements into our typeset text rather than include them in floating figure environments. It also has the advantage of encompassing all the typesetting and drawing commands in a single file that is processed with one command, with no need to keep track of a proliferation of image files and gnuplot scripts. This makes it simpler for your manuscript source to become self-documenting and easily modifiable.

This is a flexible and powerful technique with which we can combine plots generated by gnuplot with TikZ drawing commands. If we become familiar with TikZ, we will be able to arrange gnuplot graphs and PGF graphics in arbitrary ways on the page (PGF is the Portable Graphics Format, the actual drawing engine for which TikZ is a higher-level language). We don't have space here for a TikZ tutorial, but it should be possible to understand the workings of a simple example:

```
\documentclass[12pt]{article}
\usepackage{tikz}
```

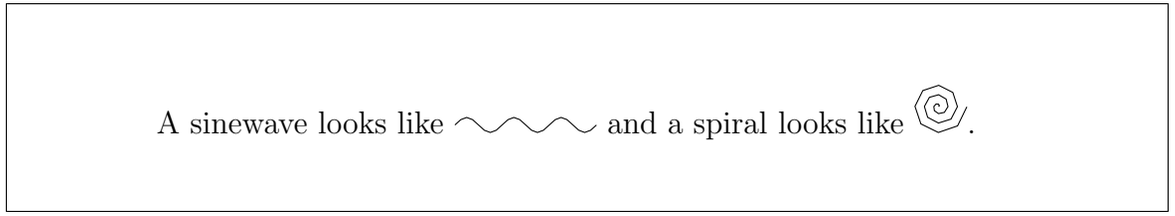
```
\usepackage{pgfplots}
\begin{document}

A sinewave looks like
  \tikz\draw[domain=0:18.84, scale=.1] plot function{sin(x)};
and a spiral looks like
  \tikz\draw[parametric, domain=0:18.84,scale=.1]
  plot function{.2*t*cos(t),.2*t*sin(t)};.

\end{document}
```

Open file

This file must be processed with an extra argument to the TeX engine that gives permission for the document to invoke an external program (in our case, gnuplot). This is for security; the concern is that malicious TeX documents might run programs without the user's knowledge, so you must turn on this ability manually. Usually, the extra argument is `--shell-escape`, but on some systems it is `--write18`. With the above file saved as `gnutikz.tex`, I processed it with the command `lualatex --shell-escape gnutikz`, and got the following result:



I got an identical result using `xelatex` in place of `lualatex`; other engines should work as well.

A few words about how this works: the pictures are placed inline with the rest of the line by the `\tikz` commands. These are followed by `\draw` with the options in square brackets. The `domain` option serves the same purpose as the [a:b] plot notation within gnuplot; the `scale` option scales the final figure by the multiplicative factor supplied; and the `parametric` option means that the following plot command is a parametric function, with `t` as the parameter and the `x` and `y` coordinates separated by a comma (we covered gnuplot parametric plotting in Chapter 1). A single command on our part is all that is required. The `function` keyword within the `\tikz` command causes LaTeX to call out to gnuplot to create tables that are subsequently read in by PGF to create the illustrations, which are then inserted into the page and typeset along with everything else. You can get a more detailed glimpse at what happens behind the scenes by looking at the auxiliary files left on the disc by the last `\tikz` command. In this case, these will be called "gnutikz.pgf-plot.gnuplot", which is the gnuplot script created by tikz, and which is run through gnuplot to create another file, called in this case "gnutikz.pgf-plot.table", which is the set of plot coordinates read in by tikz to make the little pictures. You can safely delete these auxiliary files when you are done.

TikZ/PGF can produce any type of diagram, including full-blown graphs with axes, tic marks, and so on, including LaTeX-typeset labels; but if you're making a complete graph you're probably better off running gnuplot manually and using

the techniques of the previous two sections. This is especially true if the graph is complicated or contains many elements, for in this case LaTeX processing will be significantly slower than using gnuplot to create a standalone plot file. This is because reading and parsing the resulting extremely large table will bog LaTeX down. But for placing small graph-like illustrations inline with the text, mixing them with other TikZ graphics, or combining gnuplot's talents with the flexible TikZ/PGF system to make more elaborate diagrams, the techniques described here can be uniquely powerful.

Index

LaTeX, 2
TeX, 2
TeX Live, 2, 5
TeX Users' Group, 2
TeX math
    in gnuplot plots, 10, 13, 14

backslashes, 9

Cairo, 2
convert
    the ImageMagick command, 15

enhanced text, 8
epscairo, 2
epslatex, 15
epstopdf, 15

fontspec, 12

gnuplot
    calling from LaTeX, 16

ImageMagick, 15
includegraphics, 2, 7

labels
    with TeX math, 10, 13, 14
lualatex, 5, 18

L'Hôpital's Rule, 10

mathematicians, 2

papers, 2
pdfcairo, 2, 3
pdflatex, 5
PGF, 16
physicists, 2
pngcairo, 2
PostScript, 4
Preview
    the Macintosh program, 15

quoting, 9

resolution-independence, 3

security
    in TeX, 17
shell-escape
    the TeX argument, 17
SVG, 4

tikz, 7–10, 12–14, 16, 18
    drawing commands, 18
    slow with complex plots, 18
transparency, 4

Unicode
and PostScript, 4

Web
and PDF, 4
and SVG, 4
wrapfig, 5
write-18
the TeX argument, 17

xelatex, 5