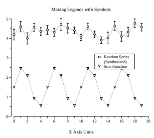
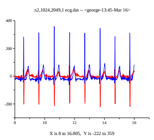
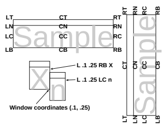
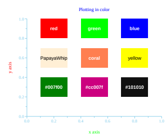
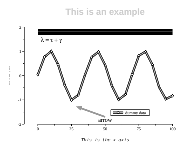
`plt` Tutorial and Cookbook

George B. Moody
Massachusetts Institute of Technology
Cambridge, Massachusetts

This copy of this book was prepared on April 27, 2005.

plt was originally written by Paul Albrecht, and is currently maintained by George Moody (george@mit.edu). The most recent version of plt, and of this book, can always be obtained from PhysioNet (http://www.physionet.org/); see appendix F, page 115, for details.

Contents

List of Figures

Acknowledgements

Thanks first to Paul Albrecht, for writing `plt` and for making it freely available in source form; and to Kim Stevenson and Marvin Appel, who wrote the *PLT USERS' GUIDE* in 1988, which for many years was the only available documentation for `plt`, passed as n^{th}-generation photocopies from user to user. Significant portions of chapters 5, 6, 7, 8, and 12 are based on Kim's and Marvin's guide, and many of the examples here are reconstructed from their work. Thanks also to the reviewers of previous drafts of this book, and especially to Joe Mietus, *pltmeister* of BIDMC.

I welcome your comments, corrections, and suggestions for improvements; please send them to `george@mit.edu`.

Chapter 1

Introduction

This book describes `plt`, a non-interactive plotting utility originally written for Unix by Paul Albrecht. `plt` can produce publication-quality 2D plots in PostScript from easily-produced text or binary data files, and can also create screen plots under the X Window System. Compared to most other software for 2D graphics, `plt` has several significant advantages:

- `plt` generates compact vector PostScript output, which can be transmitted quickly yet can be resized without introducing raster artifacts.
- `plt` works well with a wide variety of tools that create and manipulate readable text files.
- `plt` is scriptable; if you need to make 100 plots of 100 data sets, you don't need to point and click for hours.
- Complex overlays and multi-part plots are easy to make, using multiple invocations of `plt` to write to a single window or page.
- `plt` can read data from a pipe, so it can be used to observe real-time signals or the outputs of computationally intensive processes as they become available.
- `plt` imposes no fixed limits on the number of points in a plot (even the total amount of available memory is not a constraint if the data are read from a pipe and the axis limits are pre-specified).
- `plt` is free, open-source software that can be modified as needed for unique applications. (`plt` can be compiled under GNU/Linux, Mac OS X, MS-Windows, or any version of Unix.)
- `plt` is easy to pronounce (say: P-L-T) and is almost as easy to spell :-)

1.1 A brief history of `plt`

Early versions of Unix included an elegant utility called `graph`, which accepted simple text files such as:

```
1 1
2 4
3 9
4 16
```

producing from them neatly scaled and labelled 2D plots in a device-independent format. `graph` was accompanied by `plot`, which consisted of a family of interpreters (drivers) that translated `graph`'s output for a wide variety of mostly obsolete graphics terminals. I wrote a number of `plot` drivers for slightly-less obsolete devices around 1982, and Paul Albrecht wrote the first version of `plt` as a replacement for `graph` when it disappeared from the standard Unix toolkit, sometime after Version 6. (Try it out: type the numbers above into a file named `squares`, putting two numbers on each line, then run the command "`plt squares`". It's that easy!)

Early versions of `plt` used drivers I had written for the Tektronix 4010 (a vector graphics terminal with a storage display; we had a few of them around the lab), the Tektronix 4662 XY plotter that was capable of changing pens (the first color output device we had), the NEC Spinwriter (an impact printer with fine-resolution paper control, for which a special type wheel with a brass "." was available, for plotting applications such as ours), and even dumb terminals (with 80x24 resolution). During the mid-1980s, MIT's Project Athena produced the first versions of the X Window System, and Apple produced the first PostScript printer (the LaserWriter). Paul wrote X11 and PostScript drivers and integrated them with `plt`, in order to make better use of the enhanced capabilities of these environments than was possible using only the Unix `plot` API. It is these two drivers, with updates I have made since the early 1990s, that are the nucleus of `plt` today.

1.2 How to read this book

Stop! Install the current version of `plt` (see appendix F, beginning on page 115) if it is not already installed on your system, or if the version you have already was installed before April 27, 2005. Do this now, then come back here and read on.

This book attempts to be both a tutorial and a reference manual. It describes *every* `plt` option, including many that have been undocumented since `plt` was written in the mid-1980s. Although you can read this book from cover to cover, in most cases you will not need to do so in order to make high-quality plots. `plt` has a bewildering number of options, but it is very easy to make simple plots using only a small subset of these options.

If you have never used `plt` before, begin with chapter 2, *Getting Started with* `plt`, try out the examples there, and then try making a few plots of your own before continuing. Many readers will not need to go any further.

If you have used earlier versions of `plt`, you may wish to look first at appendix G, *What's New?*, for a brief summary of recent changes.

A good strategy for many users is to find in these pages an example that illustrates features you would like to include in your plot, then read the description of how the example was created. In this sense, this book is best used like a cookbook. All of the scripts and data files used to generate these examples are included in the `doc` subdirectory of the `plt` distribution (see appendix F).

Chapter 3, *Preparing Input for* `plt`, is completely new. Even if you have been using `plt` for years, you will probably find some surprises here. Chapter 4 is also new; it contains a systematic presentation of the four major coordinate systems (data, window, page, and text) used by `plt`.

Chapter 5, *Titles and Axes*, is essential reading if you have not used `plt` previously. On the other hand, almost no one will need to know everything in chapter 6, *Plotting Data*, but it is worthwhile to skim through it, so that you will know what is possible.

Chapter 7, *Plotting Two or More Data Sets Together*, will be helpful if you are trying to create a group of related plots; it offers several ways to accomplish the task of presenting multiple data sets.

Often a few well-chosen text labels can clarify a plot; chapter 8, *Labelling Your Plot*, shows how to make them. Chapter 9, *Drawing Line Segments, Arrows, and Boxes*, continues with the related subjects of adding arrows, outlined or shaded boxes, and arbitrary line segments to your plots.

Occasionally you may need to prepare a plot without one or more of the standard elements, such as axes or titles; chapter 10, *Suppressing Plot Elements*, shows how to do this without using whiteout, as well as how to include data points that fall outside of the axis ranges without inking them in.

Throughout this book are examples that use a variety of line styles, shading, color, and text fonts. Chapter 11, *Colors, Line Styles, and Fonts*, discusses how to control these aspects of your plots' appearance.

Chapter 12, *Advanced Axis Options* describes a variety of special-purpose options for fine control over axes, including how to make logarithmic axes and how to create extra labelled axis ticks.

At the end of this book, appendix A, *Color Names*, contains a complete list of the named colors that can be used for PostScript and X window output. Appendix B, *Preparing Printed Output*, contains details on creating printed plots on PostScript and other printers, and on embedding plots in LaTeX documents such as this one. Appendix D, *On-Screen Plots*, describes how to use `plt` in an X Window System environment. A few hints on writing shell scripts with `plt` appear in appendix E, *Scripting with* `plt`. Appendix F, *How to get and install* `plt`, describes how to obtain `plt` and how to compile and install it. Appendix G, *What's New?*, summarizes new features in `plt`, and includes a (short) list of known bugs. Finally, appendix H reproduces the `man` page for `plt`.

1.3 How to avoid reading this book

If you are temperamentally opposed to reading manuals, you may still be able to figure out enough of `plt` to use it from a quick look at chapter 2. The `doc` directory of the `plt` distribution contains two shell scripts:

- `xdemo.sh`, for generating and displaying the output of `plt` in an X window. Start your X server (use Cygwin/X under MS-Windows) before running `xdemo.sh`.

- `psdemo.sh`, for generating and displaying PostScript output from `plt` via `gv` (freely available for Mac OS X from Fink, for MS-Windows from Cygwin, or for GNU/Linux and Unix from SourceForge). Other PostScript viewers such as `ggv` and `gsview` can also be used (see `gvcat` in the `src` directory).

Try out one or both of these scripts to see `plt` in action, and examine the scripts to see how they work. Of course, this book is always available when you need it!

Chapter 2

Getting Started with `plt`

If you have never used `plt`, this chapter contains the essential information you need to begin making your own plots with `plt`. The first section describes how `plt` commands are constructed; don't skip this section, because it introduces important concepts and terminology that are used throughout the book. A few notes on using `plt` without an X server follow in section 2.2. The chapter concludes with a tutorial, section 2.4, in which you create a simple plot and then produce several increasingly elaborate variations that illustrate a number of the most useful features of `plt`. Be sure to work through these examples, and then try making a few plots of your own before continuing.

2.1 `plt` Essentials

The general form of a `plt` command is:

`plt` *data-spec data-file column-number ... options*

`plt` recognizes its command-line arguments by looking at the first character of each argument. If an argument begins with "`:`", it's a *data-spec*. If it begins with a digit (0-9), it's a *column-number*; if it's either "`#`" or "`%`", it's shorthand for "all of the available column-numbers, in order." If it begins with a hyphen (−), it's an *option*. If it begins with "`(`" (left parenthesis) or "`[`" (left square bracket), it's a *fontgroup specification* (not discussed further in this section; see chapter 11). If the argument begins with none of these special characters, it may be the name of a *data-file*.

A *data-spec* is a string that describes the format of the *data-file*. Most plots do not require a *data-spec* (described in section 3.4), and you may omit this argument in such cases. If you provide a *data-spec*, it must precede the *data-file* name in the command line.

The *data-file* contains the data you wish to plot, normally in text format as columns of decimal numbers (other formats are described in chapter 3). File `example1.data` illustrates the usual format:

```
0 0 0
1 1 2
2 3 1
3 5 3
4 7 2
```

This file contains three columns (from left to right: column numbers 0, 1, and 2) and five rows (from the top: row numbers 1, 2, 3, 4, and 5). The *data-file*'s name, including the suffix if any, is completely arbitrary; if the name of your *data-file* begins with one of the special characters listed above, simply prefix it with "`./`" (since "`.`" is always synonymous with the current directory, "`./anything`" is an alternate name for "`anything`"). If you omit the name of the *data-file*, `plt` reads data from its standard input. This feature allows you to pipe data from another program into `plt` without writing them to a file first.

You must specify at least one *column-number* (an integer ≥ 0) in order to plot data. The column numbers must always follow the name of the data file in the argument list (unless your data are supplied via the standard input). If you specify only one column number, `plt` assumes that the values it reads from that column are the y values, and it either reads the x values from column 0 (if the data file contains more than one column), or it generates a set of successive integers, beginning with 0, to use as the x values by default.

There are many *options* described in the following chapters. Many of them accept or require additional arguments. In most cases, `plt` can determine a reasonable default value for these arguments. If you supply a "`-`" (hyphen) on the command line in place of such an argument, `plt` will choose a default. In this book, options normally follow the *data-file* and *column-number* arguments on `plt`'s command line. This ordering is not necessary in most cases, but this convention makes it easier for humans to parse the command line.

2.2 Using `plt` without an X server

The X Window System is a standard component of GNU/Linux and virtually all current versions of Unix, including Mac OS X. A high-quality implementation of X is freely available for MS-Windows (Cygwin/X, from `http://www.cygwin.com/`); if you use MS-Windows, it will be well worth the modest effort required to download and install an X server. If you will be using `plt` with an X server, please skip ahead to section 2.3 below.

If you have not already installed `plt`, please see appendix F, beginning on page 115.

`plt` can generate PostScript plots, which can be printed directly on PostScript printers or (via `ghostscript`) on a variety of non-PostScript printers. `ghostscript` can also render PostScript on-screen, but this is most easily managed using a GUI wrapper such as `gv`, `ggv`, `mgv`, `ghostview`, or `GSView`. All of these require an X server, except that GSView can use native graphics when running on MS-Windows or OS/2. See `http://www.cs.wisc.edu/ ghost/gv/` for details on these GUI wrappers for `ghostscript`.

When using `plt` without an X server running, always send the output to `lwcat`. There are two ways to do this:

1. Send the output via a pipe, as in "`plt ... | lwcat`". (This is the recommended method.)
2. Collect the output in a file, as in "`plt ... >output.plt`", then read the file, as in "`lwcat <output.plt`". Note that the output file does not contain a PostScript prolog, so it cannot be printed directly; `lwcat` is responsible for adding the prolog.

You can easily follow along with all of the examples in this book, which work using the PostScript driver in exactly the same way as when using the X driver, if you simply add "`| lwcat -gv`" to the end of each `plt` command. (You may omit "`-gv`" if you are running within a Cygwin/bash window under MS-Windows.) When you do this, the output of `plt` is opened in a `gv` or GSView window. Using `gv` or GSView's controls, you can save the plot as a file, print it (on any printer supported by GhostScript, not just on PostScript printers), and view it at natural or magnified scales. (Drivers for a vast range of printers are included with GhostScript; see `http://www.ghostscript.com/` for a list of supported printers.)

You might reasonably wonder why `lwcat` is not part of `plt`. The reason is that the output of several `plt` commands can be concatenated and supplied to `lwcat` in a single batch to create multiple plots on a single page (see chapter 7). To do this, group the `plt` commands together within a pair of parentheses, separating them with newlines or semicolons; then send the output of the entire group to `lwcat`, either like this:

```
( plt data1 ... ; plt data2 ...; plt data3 ... ) | lwcat
```

or like this:

```
( plt data1 ... ; plt data2 ...; plt data3 ... ) >output.plt
lwcat <output.plt
```

If you are paying particularly close attention while reading the rest of this book, you may notice that `plt` accepts the option `-T lw` to specify PostScript output. If an X server is not running, this is the default, so that the `-T lw` option can be omitted. Furthermore, `lwcat` accepts the option `-gv` to specify that the output is to be opened using `gv` (GSView under MS-Windows), but this is the default if you are running `plt` and `lwcat` in a Cygwin/bash window, so that the `-gv` option can be omitted in this case. If you attempt to run `plt` and `lwcat` under MS-Windows in a DOS window (which is not recommended), or in some other way (also not recommended), you will need to use the `-gv` option explicitly.

2.3 More about options

One of the most useful options is `-h`, which causes `plt` to print a (very) concise summary of its options on the standard output. Since there are many options, the output of "`plt -h`" will scroll off of your screen, so you may wish to send it to `more` (or a similar pager):

```
plt -h | more
```

If the -h option is followed by one or more strings (which should not begin with hyphens), plt prints one-line summaries of all options beginning with those strings only.

As an alternative to supplying options and their arguments on the command line, you may write them into a *format file* that can be read by plt. This facility is particularly useful when you create a series of complex plots using some or all of the same options. You can put some of your options in a format file and supply others on the command line. Within a format file, omit the initial hyphen (−) from each option name. To use a format file, supply the option -f followed by the name of the format file, as in:

```
plt data-file 0 1 -f format-file
```

A third way to pass options to plt is within a quoted *format string*, following the -F option. Use the same syntax in a format string as in a format file. For examples of -f and -F, compare figures 5.2 and 6.1 on pages 31 and 35.

Let's see how a simple plot can be created, using example1.data. If we would like the first column to be the set of x coordinates and the third column to be the set of y coordinates, we can type:

```
plt example1.data 0 2
```

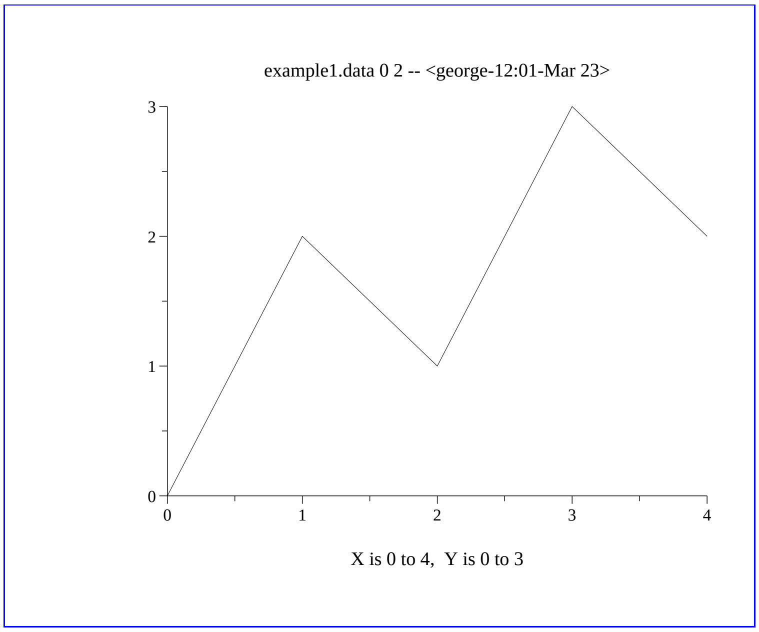
and our plot would look like figure 2.1.

Notice that in the command line, the argument "0" corresponds to the first column of example1.data, and the argument "2" to the third column. Column 0 becomes the x axis; column 2 becomes the y axis. Likewise, the argument "1" in either of these two positions would correspond to column 1 (the second column) of example1.data, and would become either the x or the y axis depending upon its position.

plt can produce both screen and printed plots. If you try the examples exactly as shown in the main part of this book, you will obtain screen plots. Although you can use other software to make and print screen dumps of these plots, this is neither the best nor the easiest way to print your plots on paper. The following sections show how you can obtain publication-quality plots on PostScript or other types of printers, with only minor changes to the plt commands needed to make the corresponding screen plots.

2.4 Tutorial: Simple Plots

Please try out the examples shown below (and throughout this book) on your own computer. You can find the necessary input files in the doc directory of the plt distribution (where you will also find the LaTeX sources for this book, and a printable PostScript version of it). For this tutorial, you will need only one file from the plt distribution, a text data file named heartrate.data, which begins:



Data

0	0	0
1	1	2
2	3	1
3	5	3
4	7	2

Figure 2.1: Produced using the command:

```
plt example1.data 0 2
```

To see how to include `plt` output as a figure in a LaTeX document such as this one, refer to appendix B.2, *Including* `plt` *figures in a LaTeX document*, page 100. The frame lines surrounding most of the figures in this book show their bounding boxes (corresponding to the borders of the `plt` window for an on-screen plot, or of the page for a printed plot.)

```
6.46389 58.0645
7.49722 59.0164
8.51389 63.3431
9.46111 65.2568
10.3806 57.2944
11.4278 57.754
```

As the excerpt above shows, `heartrate.data` contains two columns of numbers; in all, there are 297 rows (lines) in the file. The numbers on the left (in column 0) represent elapsed time in seconds, and those on the right (in column 1) indicate instantaneous heart rate in beats per minute. (See `doc/Makefile` in the `plt` distribution if you are interested in knowing more about these data.)

2.4.1 Screen and printed plots

The simplest way to plot these data is to type this command into a terminal window:

```
plt heartrate.data
```

A new window will open, containing a plot as in figure 2.2. You may be surprised to see `plt` exit to the shell prompt while the window containing the plot remains on-screen. If you run `plt` again, it will draw its output into the same window as the first. This feature allows you to create complex overlaid plots using several invocations of `plt`, as described in chapter 7, beginning on page 49. To dismiss the window, type an Esc (or `Q`, or `X`) into it. (If you wish to look at two or more plots in separate windows, open a new window for `plt` using `xpltwin`; see appendix D, beginning on page 109, for details). The same capability for creating overlays is also available when producing printed output; see appendix B, beginning on page 97, for details.

In order to illustrate the appearance of a `plt` screen plot, figure 2.2 was prepared from a screen dump. You can make printed plots of much better quality using `plt`, however, and with much less trouble than required to print a screen dump. The same plot can be printed simply by adding the optional arguments "`-T lw`" to the `plt` command, and then redirecting its output to `lwcat`, like this:

```
plt heartrate.data -T lw | lwcat
```

This command will send the plot to the default printer, producing output as in figure 2.3. Since PostScript plots are in vector rather than raster form, they can be rescaled in documents such as this without introducing artifacts, and their resolution is generally much higher than raster images despite their much smaller file sizes.

If you wish to capture the PostScript output in order to include it in a document such as this one, use `lwcat`'s `-eps` option, and redirect its standard output to a file, as in:

```
plt heartrate.data -T lw | lwcat -eps >heartrate.eps
```

See appendix B for more examples and additional details about making PostScript plots and including them in other documents.

2.4.2 Adding titles and axis labels

Now that we know how to make simple screen and printed plots, let's refine the plot a bit. First, we can provide a title for the plot, and for the x and y axes using the `-t`, `-x`, and `-y` options, like this:

```
plt heartrate.data -t "Heart rate time series" \
  -x "Time (seconds)" -y "Heart rate (beats per minute)"
```

(The "\" at the end of the first line means that the command is continued on the next line; you may enter the command on two lines, ending the first with a backslash, or you may combine both lines into one, omitting the backslash.)

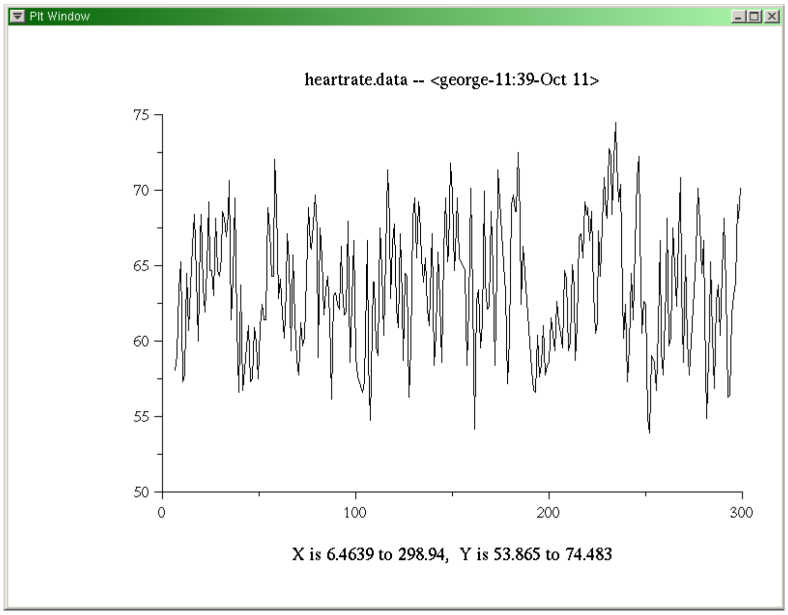


Figure 2.2: Simple screen plot

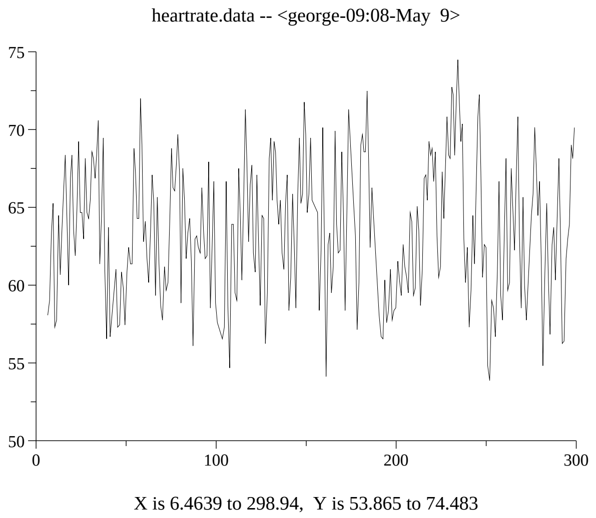


Figure 2.3: Simple printed plot

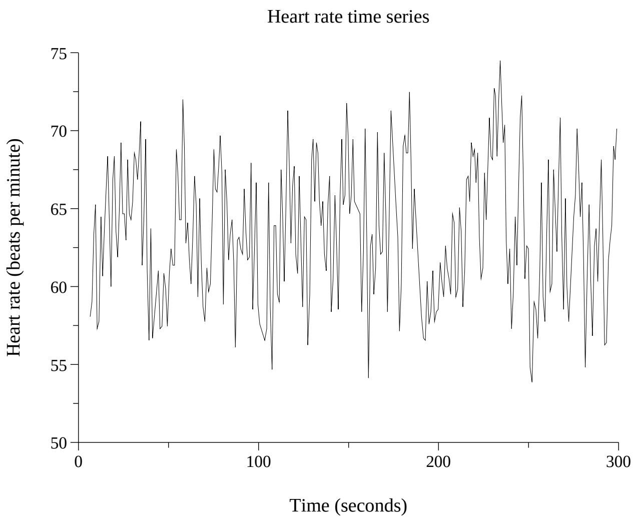


Figure 2.4: Simple plot with titles and axis labels

This command produces the neatly labelled version of the plot shown in figure 2.4. This and all of the later figures in this book were prepared as PostScript plots, using `plt`'s "`-T lw`" option and piping the output to `lwcat` as in figure 2.3. Except in the discussion of printed output in appendix B, however, the `plt` commands shown here omit "`-T lw | lwcat`", so that they can be used to make screen plots.

2.4.3 Setting up the axes

As you may have noticed, `plt` chose appropriate ranges for the x and y axes based on the ranges of x and y values in its input. Often, you may wish to specify the axis ranges, however, particularly if several plots are to be compared (in which case the plots should be prepared with the same scales and axis ranges). The next version of our plot, shown in figure 2.5, illustrates the `-xa` and `-ya` options for setting up the axes:

```
plt heartrate.data -t "Heart rate time series" \
  -x "Time (seconds)" -y "Heart rate (beats per minute)" \
  -xa 60 300 15 - 4 0 -ya 0 80 20 -g grid,sub
```

The `-xa` and `-ya` options each take an argument list containing up to six arguments. In this example, the argument list for `-xa` is "`60 300 15 - 4 0`", and that for `-ya` is "`0 80 20`" (`plt` recognizes `-g` as another option, rather than as part of `-ya`'s argument list, because `-g` consists of a hyphen followed by a letter).

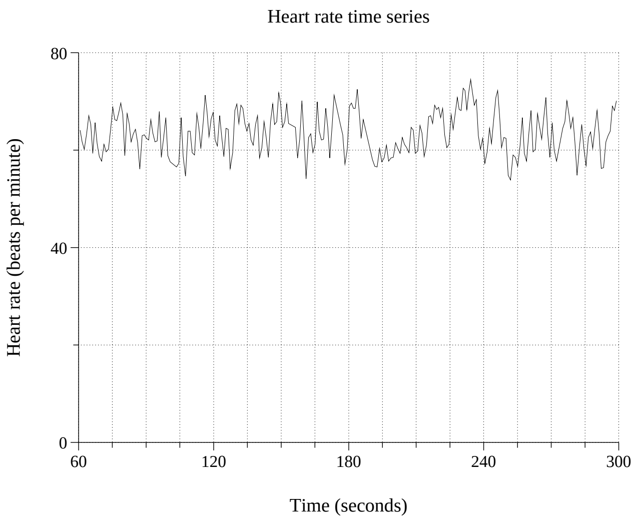


Figure 2.5: Simple plot with customized axes and grid

The first two arguments following `-xa` and `-ya` specify the x and y axis ranges. Notice that the x axis range (60 to 300) excludes a number of the points at the beginning of the data file. If you try this command, `plt` will warn you (by printing a message in your terminal emulator window) that points were excluded from the plot; note that there is no indication of this on the plot itself, however.

The third argument following each of `-xa` and `-ya` specifies the interval between axis ticks (in the units of the data). We have chosen 15 units for the x axis tick interval, since the x units are seconds and this allows us to arrange for a numbered tick at the beginning of each minute (the `4` following `-xa` specifies that every fourth tick is to be numbered).

The "`-`" between the `15` and the `4` is an example of a *default argument*. In most cases, where `plt` expects an argument, it will choose a reasonable default if you provide "`-`" instead of a specific value. The value that "`-`" replaces in this case is a format string that can be used to control how the numbered x axis ticks are printed. Note that, as in the argument list for `-ya` in this example, it is also usually acceptable to omit arguments at the end of a list if `plt`'s defaults are acceptable to you. For further details on using the `-xa` and `-ya` options, see chapter 5.

The `-g` option and its argument (`grid,sub`) specifies that `plt` should draw gridlines across the plot at every axis tick. See chapter 12 for details about using `-g` to control the appearance of the grid.

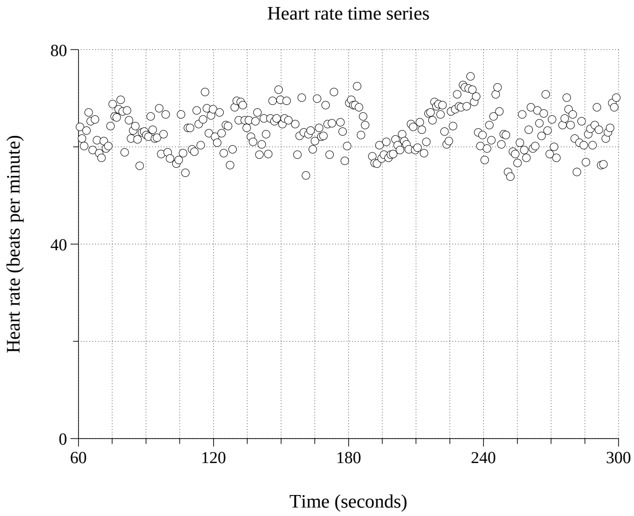


Figure 2.6: Scatter plot

2.4.4 A simple scatter plot

In the figures above, the data are plotted by connecting the points with straight line segments. This is `plt`'s default plot style, and for many plots, it is adequate. For `heartrate.dat`, however, a scatter plot may be a more appropriate format. The next version of our plot adds the option "`-p 0,1Scircle`" to the `plt` command in order to create a scatter plot, as shown in figure 2.6:

```
plt heartrate.data -t "Heart rate time series" \
  -x "Time (seconds)" -y "Heart rate (beats per minute)" \
  -xa 60 300 15 - 4 0 -ya 0 80 20 -g grid,sub \
  -p 0,1Scircle
```

The use of the `-p` option to specify a plot style is described in chapter 6; `plt` offers a wide range of styles. The argument "`0,1Scircle`" specifies that columns 0 and 1 of the data are to be used to make a plot in style `S` (a scatter plot) using an open circle to mark each data point.

2.4.5 Using color

We conclude this tutorial by adding a *transient fontgroup specification* (see chapter 11) that has the effect of modifying the appearance of the scatter plot symbols. To do this, we append "`(P/2,Cblue)`" to the argument of the `-p` option, like this:

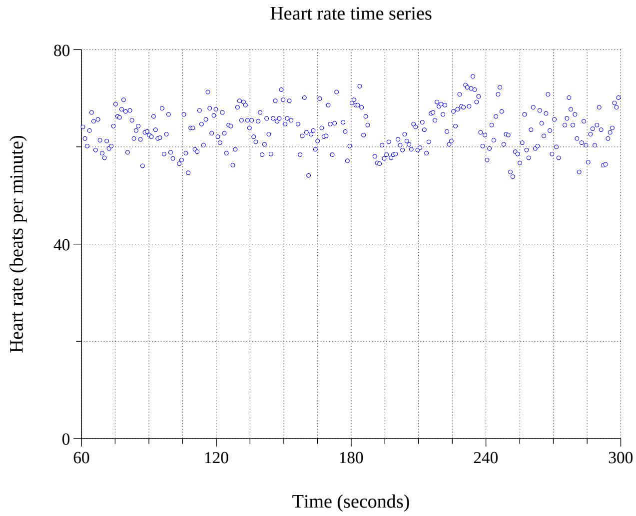


Figure 2.7: Scatter plot in color

```
plt heartrate.data -t "Heart rate time series" \
  -x "Time (seconds)" -y "Heart rate (beats per minute)" \
  -xa 60 300 15 - 4 0 -ya 0 80 20 -g grid,sub \
  -p "0,1Scircle(P/2,Cblue)"
```

(Since the shell (command interpreter) normally treats parentheses as special characters, it is necessary to enclose the entire argument of `-p` in quotation marks, in order to pass it as shown to `plt` without shell interpretation.)

Fontgroup modifications provide an extremely powerful and flexible way to control the appearance of a plot. In this example, "`P/2`" reduces the size of the circles by a factor of 2, and "`Cblue`" causes them to be drawn in blue. (A complete list of color names is included in appendix A.)

2.5 Plotting an image

The `plt` package includes a tiny program named `imageplt` that can be used together with `plt` to plot a greyscale image, as shown in figure 2.8. The input to `imageplt` should be a list of grey levels (0 = black, 1 = white) for each pixel in the image, beginning at the lower left corner, going up each column, then continuing from the bottom of the next column, etc. For example, suppose the input file `image.data` contains:

```
1 .9 .8 .7 .6
0 0 0 0 0
1 1 1 1 1
```

These data can be plotted as a 3x5 image to produce figure 2.8. For further details, see Appendix H.

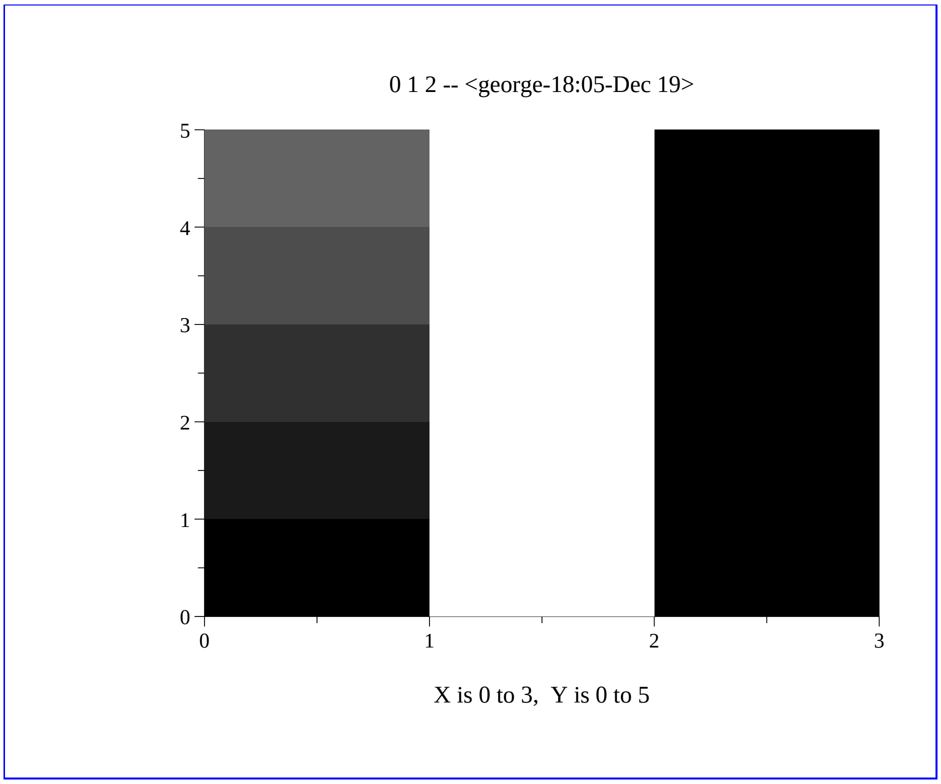


Figure 2.8: Produced using the command:

```
imageplt -d 5 3 image.data | plt 0 1 2 -pc
```

2.6 Plotting a function of a single variable

`plt` plots *(x, y)* coordinate pairs, but often you may wish to plot a continuous function of *x* expressed symbolically. This can be done easily using two small programs, `ftable` and `pltf`, both of which are included with `plt`.

Both of these programs accept a symbolic function definition (as the first command-line argument) and up to three optional arguments: the lower and upper bounds for the independent variable (which is always `x`), and the `x`-increment.

`ftable` produces a script that generates a table of coordinate pairs when processed by the standard `bc` utility. This table can then be used as input to `plt`. For example, the command:

```
ftable 'x^2 - 4*x + 7' 0 5 .1 | bc -l
```

produces on its standard output a table of values of the function $f(x) = x^2 - 4x + 7$ for values of x between 0 and 5, with a step size of .1 in x. Note that the independent variable in the function is always `x`, and that multiplication must be indicated explicitly using `*`. In this example, the output begins like this:

```
0 7
.1 6.61
.2 6.24
.3 5.89
.4 5.56
.5 5.25
.6 4.96
.7 4.69
```

The `bc` utility is an arbitrary-precision calculator included in all versions of Unix and GNU/Linux, and available for Mac OS X (from `http://fink.sourceforge.net/`) and for MS-Windows (as part of the free Cygwin package from `http://www.cygwin.com/`). See the documentation for `bc` for details on the function syntax.

The shell script `pltf` is included in the `misc` directory of the `plt` distribution. `pltf` accepts the same arguments as `ftable` (the function, the lower and upper bounds for `x`, and the `x`-increment), but it invokes `bc` and `plt` to produce a neatly labelled plot of your function, as illustrated in figure 2.9:

```
pltf 's(40*x)*s(3*x)' 0 5 .01
```

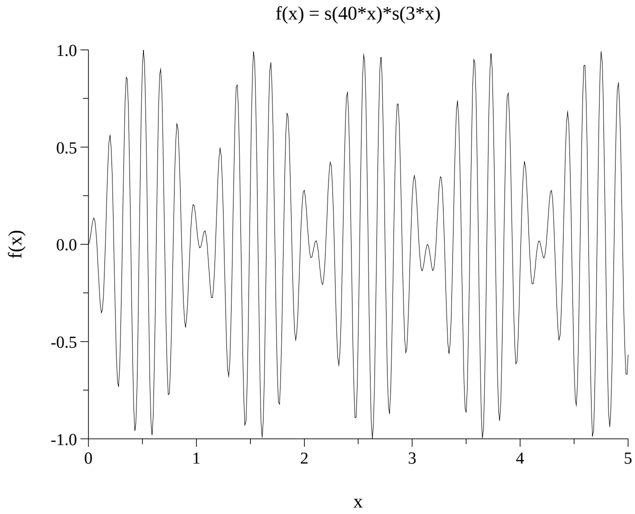


Figure 2.9: Function plot made using `pltf`

Chapter 3

Preparing Input for `plt`

Conceptually, all data presented to `plt` must be organized in rows and columns. Columns are numbered beginning with zero, and each column contains values for a variable that can be used as an abscissa (x coordinate), ordinate (y coordinate), or (with appropriate options discussed later in this book) a grey level, color, or other plot attributes. Rows are numbered beginning with one, and each row contains a value for each column. Within a data file, values are always arranged in row-major order (all elements of row 1, followed by all elements of row 2, etc.).

`plt` can read data from a file or from a pipe. A command-line argument that is not otherwise recognized by `plt` is assumed to be the name of a data file. Any legal file name is acceptable; if it begins with a character that would cause `plt` to interpret it as an option, simply prefix the name with "`./`". If no data file is named, `plt` reads data from the standard input.

3.1 Text data files

Usually, data must be in text form in order for `plt` to read them. Each non-empty, non-comment line (row) in the input should contain a value for each column that will be plotted; any additional values or other extra text at the end of a row will be ignored. Columns can be separated by any number of spaces or tabs. Commas and single or double quotation marks can also be used as column separators with current versions of `plt`, though not with older versions. It is not necessary to line up the values in each row. There may also be spaces or tabs at the beginning of a line, and these will also be ignored. For example, here is `squares.data`, a text data file that can be read by `plt`:

```
# This is a text data file.  It doesn't need to begin with a
# comment, although this one does.  Comments and empty lines
# are ignored in text data files.

-1 1 # This is row 1.  It contains two columns.
0    0 # This is row 2.  Columns don't need to be lined up neatly.
  1   1 # Row 3.  Whitespace separates columns and is otherwise ignored.
1.5 2.25# Row 4. Most common formats for base 10 numbers are acceptable.
# Note, however, that only "." can be used to separate
# the integer and fractional parts ("," is treated as a
# column separator).

2 4 # Row 5 (the empty and comment lines aren't counted.)
2.5 6.25 8  # The "8" is ignored, since row 1 has only 2 columns.
3 9
4,16 # Comma-separated value (CSV) format is acceptable.
"5","25"# Quoted CSV format is also acceptable.
6 3.6e1 # Standard notation for floating-point numbers is acceptable.
7 490e-1# Here's another, with a negative exponent.
8 64.
9 +81.0 # The "+" is redundant, but acceptable.
10 1e2

# There can be more comments such as these at the end ....
```

3.2 Comma-separated value (CSV) files

Many programs record data in comma-separated value format. Current versions of `plt` can read most CSV files. It is still necessary for each row to be on a separate line, but commas are acceptable as whitespace equivalents for separating data values within a single row (line). If your CSV file contains quoted values, `plt` ignores the (single or double) quotation marks.

3.3 Binary data files

`plt` can also read some types of binary data files. You can create files of these types by writing the data as a sequence of `short` integer, `float`, or `double` format numbers. These types may not be mixed in a single data file. The sizes of these data types and their byte ordering rules are architecture-dependent, so that binary data files, unlike text data files, are not necessarily portable between computers of different types.

The structure of binary data files is similar to that of text data files, in that data are (conceptually) organized in rows and columns. If your file contains values for N variables (columns), each row must contain N values, one for each column. The data file does not contain any row boundary markers; you must specify the number of columns and the data type using either an embedded two-byte descriptive header at the beginning of the data file, or an appropriate *data specification* on the `plt` command line, so that the input can be parsed correctly.

A binary data file can be most easily used by `plt` if its first two bytes describe its format. To do this, the first byte should contain the number of bytes per data value (on most current CPUs, this is 2 for `short`, 4 for `float`, or 8 for `double` values), and the second byte should contain the number of columns per row. These header bytes should be followed immediately by the data. `plt` can recognize binary files of this type because text files should contain printing characters and whitespace only (the byte values 2, 4, and 8 correspond to non-printing, non-whitespace control characters).

The two-byte header is optional. If it is omitted, however, you must use a data specification (see the next section) in order to advise `plt` of the format of your data. Binary data files to be used as input to `plt` should not contain any information other than the optional header and the data themselves, although it is possible to skip over an embedded prolog or epilog in most cases.

3.4 Data specifications

Use a *data specification* to advise `plt` of the format of your data file, or to select rows from the file that you wish to plot. Any `plt` command may include a data specification. A data specification is required in order to use a binary data file that lacks a header. The data specification is a string beginning with "`:`", and always precedes the name of the data file in the `plt` command line. Up to four optional specifiers, separated by commas, can follow the "`:`", as in:

```
plt :s2,1024,2049,1 ecg.dat \
    -cz 8 .00781 -F"p 0,1n(Cred) 0,2n(Cblue)"
```

This command produces Figure 3.1 on page 23. The data specification in this example is "`:s2,1024,2049,1`", the data file is `ecg.dat`, the `-cz` option is described in the next section, and the remaining arguments are options described later in this book. All four optional specifiers are shown in this example; in order, they are:

format This specifier is used for binary data files only. It is a letter that indicates the data type, followed by a number that indicates the number of values (columns) per row. Use one of "`s`", "`f`", or "`d`" to specify `short` (integer), `float`, or `double` format data respectively. In the example, *format* is "`s2`", indicating that the input is a binary data file containing `short` integer data in 2 columns. If *format* is omitted, `plt` assumes the input is a text data file.

min-row If present, `plt` excludes rows with smaller row numbers. In the example, *min-row* is 1024.

max-row If present, `plt` excludes rows with equal or greater row numbers. In the example, *max-row* is 2049.

dec If present, `plt` includes only one of every *dec* rows, beginning with *min-row*. In the example, *dec* is 1 (all rows are plotted); this is the default, and could have been omitted.

Omitted specifiers are replaced with default values; the defaults are to include all rows. For example, the specification "`:100`" excludes rows 1-99 and includes all others; the specification "`:2,,2`" excludes all odd-numbered rows, and "`:1,,2`" (or "`:,,2`") excludes all even-numbered rows.

If your data file contains an embedded prolog of a type other than the two-byte header used by `plt`, you may be able to skip over it using an appropriate value of *min-row* in your data specification. If the file contains binary data, and if the length of the prolog is not a multiple of the size of a row in bytes, you will need to use another method; the Unix utility `dd` may be useful for this purpose, as in:

`dd ibs=1 skip=`*nbytes* <*data-file* | `plt ...`

3.5 Generating abscissas automatically

If `plt`'s argument list includes only one *column-number*, `plt` reads the y values (the ordinates) from that column of the data file. In this case, if the data file contains more than one column, `plt` reads the x values (the abscissas) from column 0. If the data file contains only one column, `plt` automatically supplies the abscissas. The automatically generated abscissas can be referred to as column 0, and the ordinates become column 1.

If you have multi-column data that lacks abscissas, you can force `plt` to generate a column of abscissas using the `-cz` (column zero) option. When you use `-cz`, your data columns are renumbered, so that the first one (which would normally be called column 0) becomes column 1, the second becomes column 2, etc.

When `plt` supplies the x values, it creates a column of x values starting with x_{from} (by default, 0), and incrementing by x_{incr} (by default, 1) for each subsequent value of x. The `-cz` option can be used to change these values, as follows:

`-cz` x_{from} x_{incr}

This feature is illustrated in figure 3.1. The data file `ecg.dat` contains samples of two ECG signals taken at 128 samples per signal per second, hence x_{incr} is set to $\frac{1}{128}$, or 0.00781, so that the x units on the plot are seconds. Since the plot begins with row 1024 ($8 \cdot 128$), x_{from} is set to 8, so that the x axis marks indicate the elapsed time in seconds from the beginning of the data file.

3.6 Format files

As noted earlier, `plt`'s command-line options can also be collected into *format files*, which are read by `plt` if named following the `-f` option. The names of format files, as for data files, are completely arbitrary (in this book, format files have the suffix `.format`, but this is only for clarity).

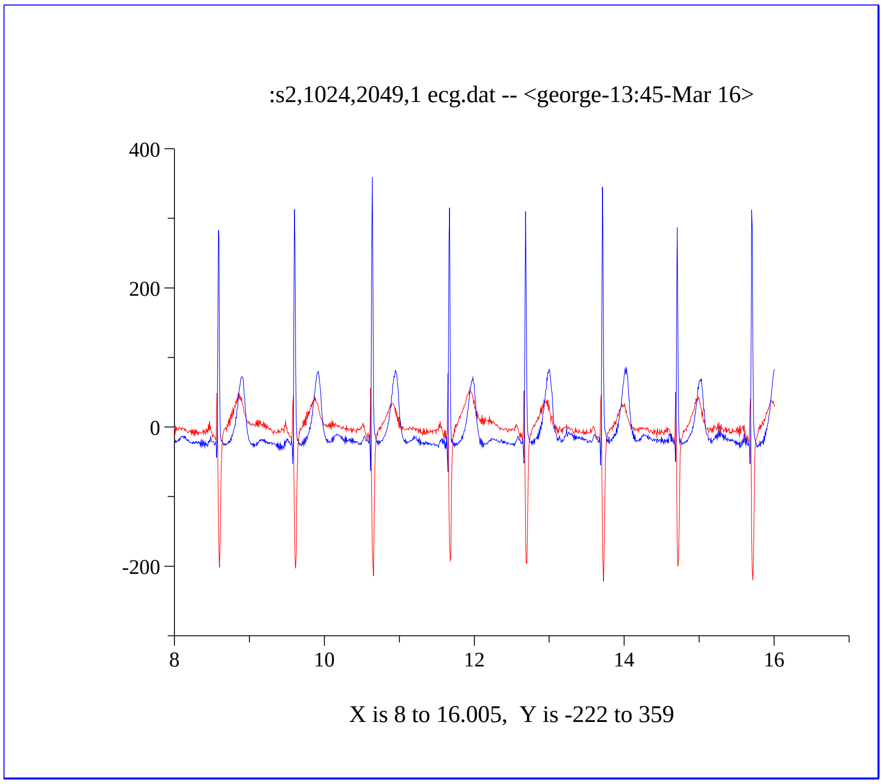


Figure 3.1: Produced using the command:

```
plt :s2,1024,2049,1 ecg.dat \
    -cz 8 .00781 -F"p 0,1n(Cred) 0,2n(Cblue)"
```

Format files are text files. Like data files, they may contain empty lines, and any line beginning with '#' is taken to be a comment and is ignored.

The active ingredients in format files are `plt` options and their arguments. In a format file, you must omit the initial hyphen ('-') from each option named. See chapter 5 for examples of figures made using command-line arguments with hyphenated options, and using a format file containing unhyphenated options.

3.7 String arrays

In chapter 6, you will encounter `plt` options that make use of yet another type of input, namely arrays of strings. String arrays can be supplied to `plt` using the `-fs`, `-ts`, and `-tf` options. These options take arguments as follows:

> `-fs "`$string_0$ $string_1$ ...`"`
> `-ts "`$string_0$ $string_1$ ...`"` *tbc*
> `-tf` *file tbc*

The string argument of `-fs` and `-ts` is treated as an array of strings (words) separated by white space. The *file* argument of `-tf` is the name of a file that contains an array of strings separated by newlines (the strings may include spaces or tabs in this case). The `-fs` option defines the *fontgroup string array*; the `-ts` and `-tf` options define the *text string array*. In each case, strings within a string array are referred to by their index numbers, and (as for the column numbers in the data file) the index number of the first string in a string array is 0, not 1.

Entries in the text string array can be plotted in the same manner as data points, by using other options described in chapter 6, together with a data file that specifies the data coordinates and the index numbers of the strings within the text string array. The *tbc* argument of `-tf` and `-ts` is optional; if provided, it specifies a *text box coordinate* used to determine how the strings are placed relative to the coordinates specified by the data file (see chapter 4). If *tbc* is omitted, `plt` centers the strings on the specified coordinates.

The fontgroup string array associated with `-fs` has a different purpose; using the appropriate options, these strings can be used to set the font, line style (solid, dotted, etc.), or plotting color. Examples of the use of these options appear in chapter 6.

Chapter 4

Coordinate Systems

Using appropriate `plt` options, you can place labels anywhere in a plot, and you can place plots anywhere on a page. These `plt` options use four coordinate systems to specify position.

First is the *data coordinate system*: the coordinates defined by your plot's axes. The ranges of the data coordinates depend on the ranges of x and y values, or on the ranges you specify using the `-xa` and `-ya` (or `-X` and `-Y`) options. The lower left corner of the data coordinate system is (x_{min}, y_{min}), and the upper right corner is (x_{max}, y_{max}).

Second is the *window coordinate system*. Window coordinates, (xw, yw), always range from $(0, 0)$ to $(1, 1)$. The window coordinates $(0, 0)$ correspond to the data coordinates (x_{min}, y_{min}), and the window coordinates $(1, 1)$ map to the data coordinates (x_{max}, y_{max}).

Third is the *page coordinate system*. Page coordinates, (xp, yp), always range from $(0, 0)$, at the lower left corner of the page, to $(1, 1)$ at the upper right corner. Using the `-W` option described later on, you can specify a rectangle in page coordinates in which the plot is to be drawn; this option thus allows you to draw multiple plots in any desired layout on a page. When you make a screen plot, the "page" may be the entire screen, or it may be an X window. When you make a printable plot, the "page" may be the entire sheet of paper, or the printable area (generally less than the full sheet), or it may represent a PostScript bounding box for a plot that can be incorporated into another document such as this book.

Figure 4.1 illustrates the relationships among data, window, and page coordinates. Note that it is always possible to refer to data, window, or page coordinates outside of the ranges of these coordinate systems. Depending on the `plt` options you have chosen, and in some cases on the output device (screen or printer), out-of-bounds elements may or may not appear in the output.

A fourth coordinate system, called *text box coordinates*, is used by `plt` to allow fine control over the positioning of text on plots.

`plt` places an imaginary *text box* around each string it prints. Descenders (the lower portions of "g", "j", "p", "q", "y", and ",") fall outside of the box, but each text box includes extra space above the top of the characters (30% to 40% of the basic height of the text) for any descenders from the line above. Text box coordinates are

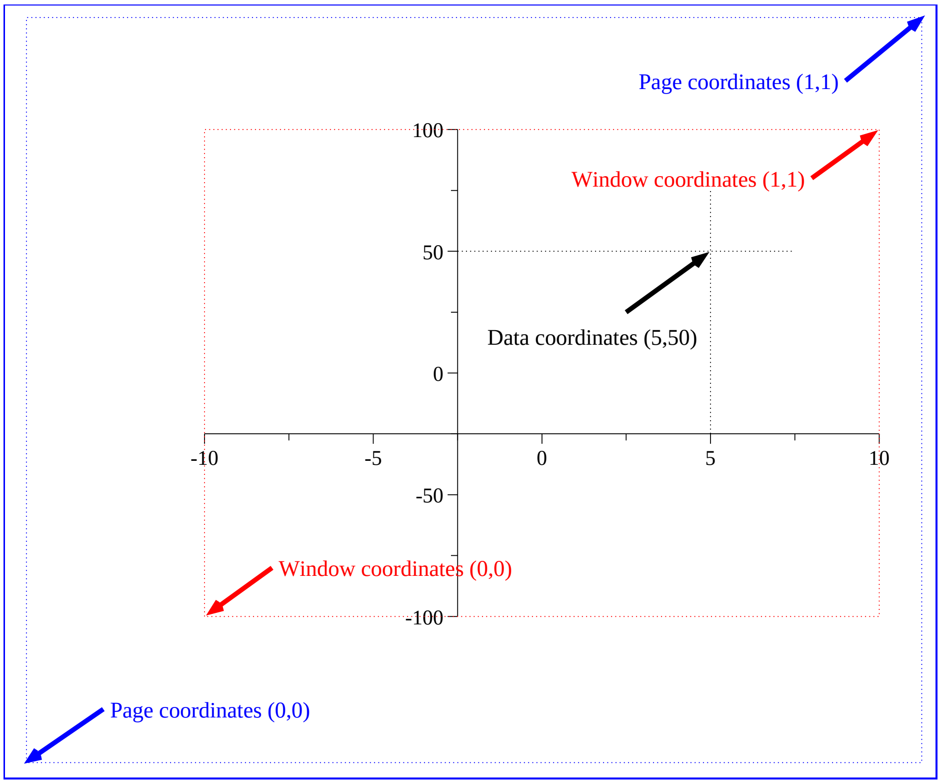


Figure 4.1: Data, window, and page coordinates

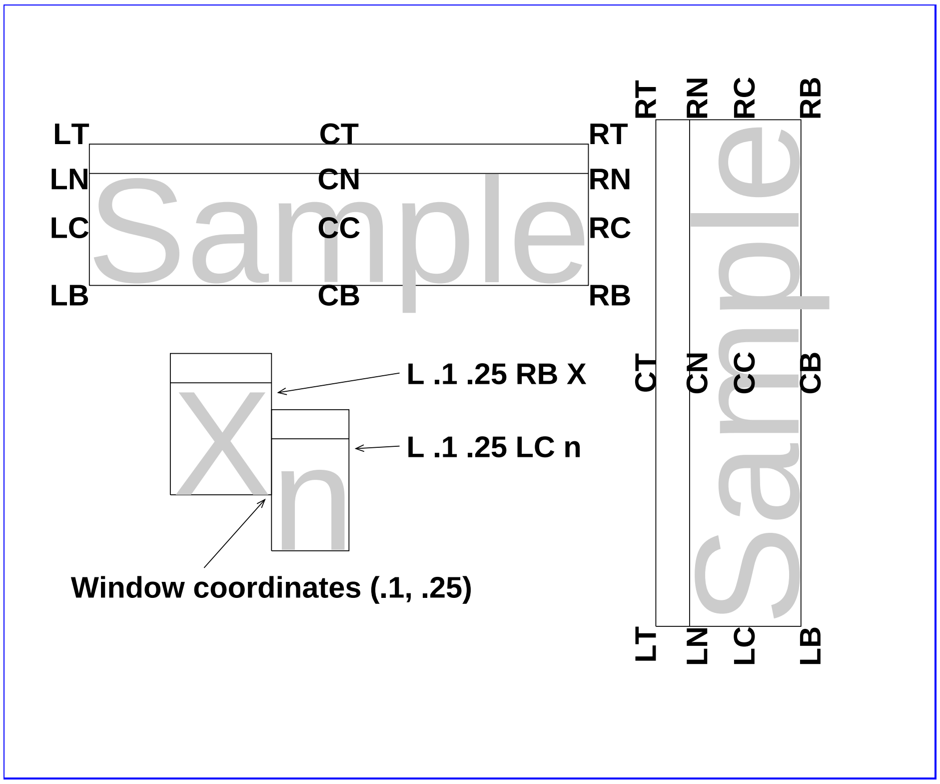


Figure 4.2: Text box coordinates. Note that text box coordinates are always oriented left-right and top-bottom with respect to the text, and not necessarily with respect to the page. At lower left is an example of how to form subscripts using `plt`'s `-L` option (see chapter 8). As shown, the right bottom (`RB`) point of the `X` and the left center (`LC`) of the subscript `n` are both placed at the same window coordinates (.1,.25) to achieve the desired alignment.

symbolic and discrete rather than numeric and continuous. Each text box has twelve points defined as shown in figure 4.2. Thus point `CT` is at the center of the top side of the text box, `RB` is the lower right corner, etc.

When you tell `plt` to print a string at (x, y), `plt` usually places text box point `CC` at (x, y), thus centering the string on (x, y). By naming the appropriate text box point as an argument to any of the options described in chapter 8, *Labelling Your Plot*, you can choose instead to place any of the four corners, the center of any side, or any of the other defined points of the text box at (x, y).

Chapter 5

Titles and Axes

By now you should know how to make a simple plot of data as in the previous examples. You are now ready to label your plot. Using the `-x` and `-y` options, you can label the x and y axes respectively. Using the `-t` option, you can give your plot a title. Figure 5.1 demonstrates the use of these options.

```
plt example1.data 0 2  -t "Time vs Amp" \
    -x "time in seconds" -y "amplitude in cm"
```

Note that the data file name and the column numbers should immediately follow "`plt`".

With the `-xa` (x axis) and `-ya` (y axis) options, you can specify *min* and *max* (the endpoints of your axes), *tick* (how often you want tick marks to appear), *fmt* (the format in which to print the numbers, e.g., %.3f, %.2e), *tskip* (how often to skip when labelling the ticks, and *cross* (where you want the axis you are specifying to cross the other axis, in units of the other axis). If you do not need to change the default ticks and axis crossing points, the `-X` and `-Y` options accept axis endpoint arguments only. These options take their arguments as follows:

`-xa` *xmin xmax tick fmt tskip ycross*
`-ya` *ymin ymax tick fmt tskip xcross*

`-X` *xmin xmax*
`-Y` *xmin xmax*

Figure 5.2, on page 31, demonstrates how `-xa` and `-ya` are used, and also the use of the `-f` option with a format file. The format file, named `example3.format`, looks like this:

```
t Time vs Amp
x time in seconds
y amplitude in cm
xa 0 5 .5 - 2 0
ya 0 5 .5 - 2 0
```

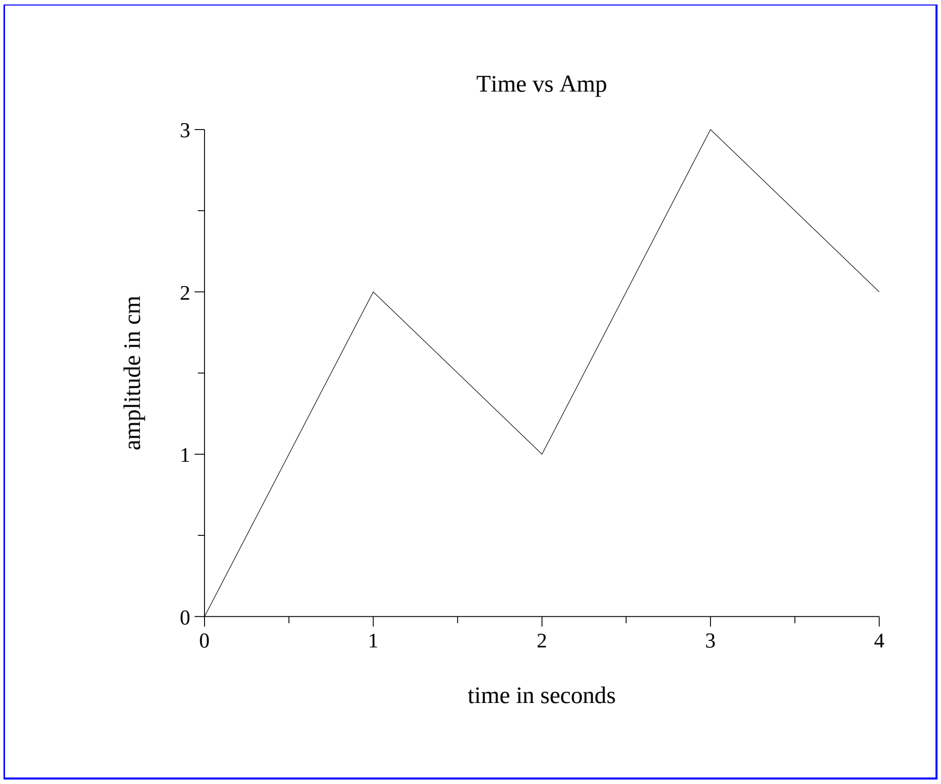


Figure 5.1: Produced using the command:

```
plt example1.data 0 2  -t "Time vs Amp" \
    -x "time in seconds" -y "amplitude in cm"
```

As for the data file, the name of the format file, including the suffix if any, is completely arbitrary. Note that each option within the format file (in this example, `t`, `x`, `y`, `xa`, and `ya`) must be given without an initial "–", but that a "–" in place of an option argument will cause `plt` to choose its own "suitable value" for that argument. Either a number based on the size of the plot and data limits will be chosen, or `plt` will use a default value in place of the "–" argument.

`plt` offers a large number of options that give you control over virtually every aspect of how axes are drawn and labelled; these are described in chapter 12, beginning on page 85.

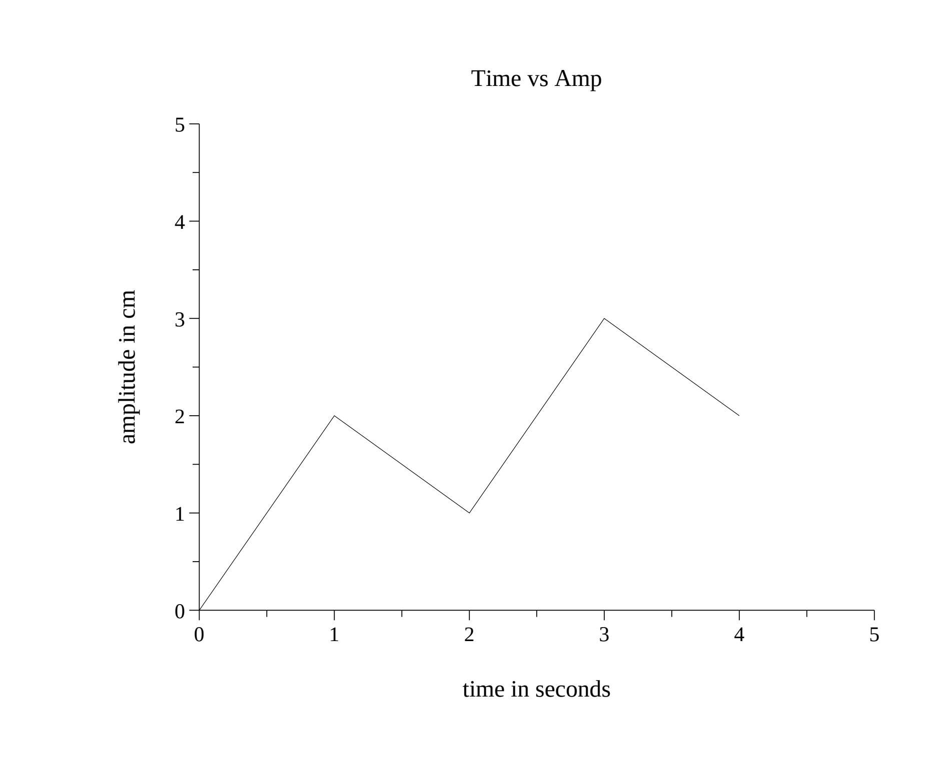


Figure 5.2: Produced using the command:

```
plt example1.data 0 2 -f example3.format
```

5.1 "Quickplot" mode

In normal operation, `plt` performs two passes over its input data. During the first pass, it reads the data from a file or from its standard input, allocates memory sufficient to keep the data to be plotted, and determines suitable axis ranges; during the second pass, `plt` renders the plot. When making a screen plot, the intermediate steps are not visible; for efficiency, the `plt` window is not (re)drawn until the entire image has been prepared in memory.

If, however, you specify both x and y axis ranges using any of the `-xa`, `-ya`, `-X`, and `-Y` options described above, *and* if you are making only one plot (i.e., not using any of the methods described in chapter 7 to plot multiple data sets on a single set of axes), then `plt` runs in *quickplot* mode. In quickplot mode, the data are read and plotted immediately without buffering, so that for large data sets the memory requirements and the time needed to render the plot are significantly reduced. Quickplot mode allows you to plot data sets that are larger than the total amount of available memory. Furthermore, under Unix, GNU/Linux, and Mac OS X, quickplot mode permits progressive display of screen plots, so that you do not need to wait for `plt` to read an entire multi-megabyte input file before seeing any output. If the data are provided by a pipe from another process, `plt` can render them in real time (depending on your hardware speed, at rates up to 50 frames per second, or even faster if you change the symbol `QPFREQ` in `xw.c` and recompile `plt`; see appendix F).

Figure 5.3 illustrates the final state of a plot of 25,000 points from a Hénon series. The command used to generate the figure was:

```
henon | plt % -p s. -X -1.5 1.5 -Y -.5 .5 -t "Henon Attractor"
```

You can see quickplot mode in action if you compile `henon` from the source (a file named `henon.c`, provided in the `doc` directory of the `plt` distribution) and then run the command above. Delays are included in `henon.c` so that you will be able to see what happens even with a fast CPU. Over a period of several seconds, the picture of the Hénon attractor emerges.

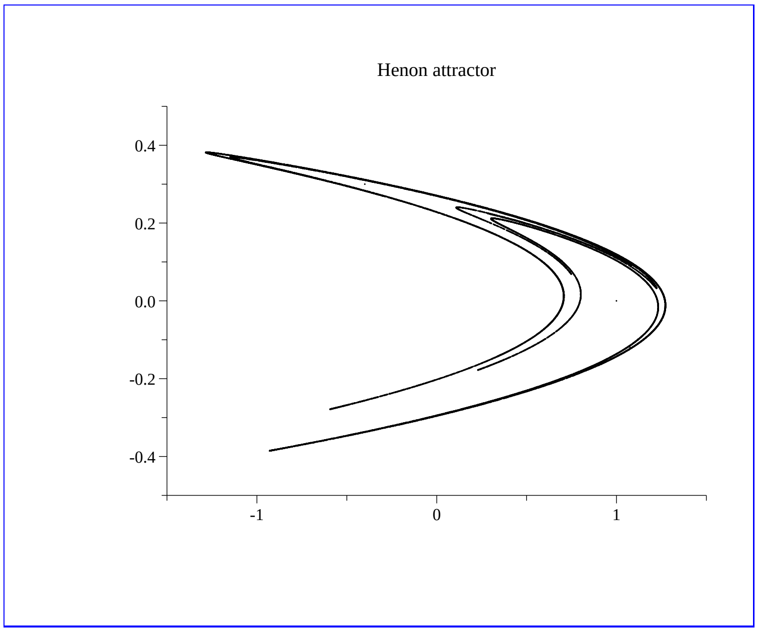


Figure 5.3: Hénon attractor, produced in *quickplot* mode

.

Chapter 6

Plotting Data

Most of the previous examples have illustrated how two columns of data can be plotted as a connected set of line segments. This is `plt`'s default style for plotting data, but many other styles are supported using the `-p` option to choose a *plotstyle*. For example, data can be plotted using points, characters, strings, or symbols (scatter plots); with error bars; or using lines of various widths, grey levels, colors, or patterns (solid, dotted, etc.).

To specify a plotstyle, we can use the `-p` option, followed by sub-options that control how the points are plotted. Each sub-option can also have an optional argument following it. This argument can change the font, point size, color, gray scale, line width, or line style (solid, dotted, dashed, etc.) of the current plot. (If you want to use those control options that enable you to make font changes, you will need to read and understand chapter 11, *Colors, Line Styles, and Fonts,* beginning on page 73.)

Each plotstyle "takes" (consumes) one or more data columns. If you use more than one plotstyle, you must include enough data column numbers for all of the plotstyles in the `plt` command line. Note that you may repeat data column numbers if necessary to satisfy the requirements of the plotstyles you select; see the examples below to see how this can be done.

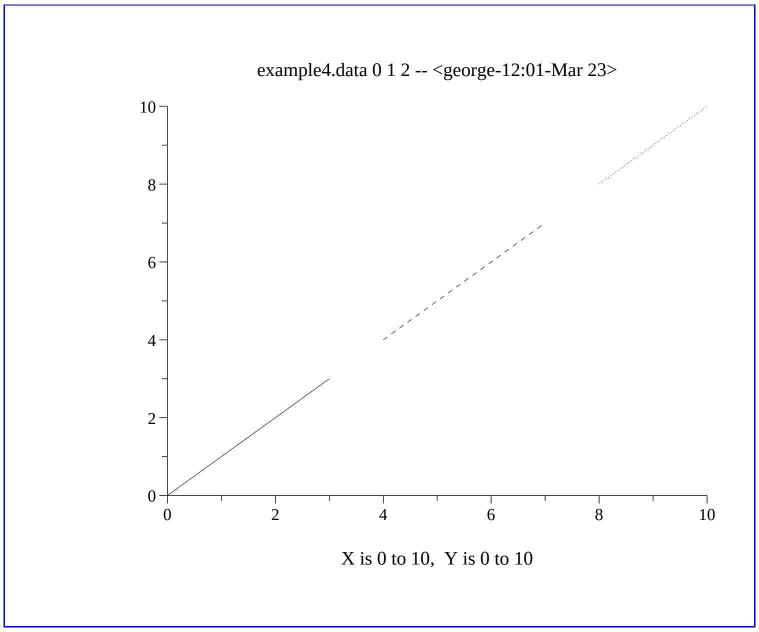
If you have not used `plt` previously, it is probably a bad idea to read through this chapter from beginning to end, because the usage of the plotstyles (especially the first one listed below, `c`) is mind-bogglingly convoluted. A better strategy is to begin by studying the examples in this chapter; find one or more that illustrate features you would like to use in your own plots, then see which plotstyles were used to obtain these features.

The sub-options to `-p` are:

`c` Takes three data columns (x, y, and c). Each c value tells `plt` what to do with the corresponding x and y values. The values for c can be:

`0`	continue (draw) path to (x, y)
`1`	move to (x, y) without drawing (i.e., with the "pen" up), then put the pen down (begin a new path)
`2`	put a dot at (x, y)
`3`	put a small box at (x, y)
`9`	paint the path (usually done as the default when a new path is begun without specifying what to do with the old one)
`10`	continue to (x, y), close path and fill the inside with grey level specified by the argument to the `c` sub-option
`11`	continue to (x, y), close path and fill the inside with grey level specified by the argument to the `c` sub-option; then draw a black border
`12 – 13`	continue to (x, y), close path and fill the inside with grey level specified by $c - 12$ (12.0 = black, 13.0 = white)
`14 – 15`	continue to (x, y), close path and fill the inside with grey level specified by $c - 14$ (14.0 = black, 15.0 = white); then draw a black border
`20`	(requires `-fs` option, see section 3.7, *String arrays*, page 24) change font to that specified by string number x from the `-fs` string array
`21`	change point size to x
`22`	change line width to x
`23`	(requires `-fs` option, see section 3.7) change line style to that specified by string number x from the `-fs` string array; legal line styles are "`solid`", "`dotted`", "`shortdashed`", "`dotdashed`", and "`longdashed`". Note that y is ignored (see figure 6.1).
`24`	change grey level to x (0 = black, 1 = white); y is ignored.
`25`	(requires `-fs` option, see section 3.7) change color to that specified by string number x from the fontgroup string array (see appendix A for details on how to specify colors by name; also, note that y is ignored)
`30 – 39`	put symbol number $c - 30$ at (x, y) (see figure 6.2)
`100, 101, ...`	(requires `-tf` or `-ts` option, see section 3.7) put string number *control-100* from the text string array at (x, y)

`C` Takes two columns, x and y, assumed to be the vertices of a polygon. The polygon is drawn by connecting consecutive vertices and by connecting the last vertex to the first, and then it is filled in the current color (default: black). Nonconvex and self-intersecting polygons are filled properly. See figure 6.3, page 39.



Data

```
0 0 0
3 3 0
1 7 23
4 4 0
7 7 0
2 6 23
8 8 0
10 10 0
```

Figure 6.1: Produced using the command:

```
plt example4.data 0 1 2 -F"\
    fs helvetica longdashed dotted\
    p c"
```

The two lines in the data file (at right, above) in which the third column (c) is 23 change the line styles to string x (the number specified in the first column) from the fontgroup string array. The fontgroup string array is defined in the command line above (string 0 is "`helvetica`", not used in this example; string 1 is "`longdashed''`; and string 2 is "`dotted`"). Thus when `plt` reads `1 7 23`, the line style changes from the default ("`solid`") to the style specified by string 1 ("tt longdashed"; the 7 is ignored), and when it reads `2 6 23`, the line style becomes "`dotted`" (and the 6 is similarly ignored).

The line segments are not connected, because each non-zero c value causes the previous segment to be drawn, and a new segment begins at the next (x, y) pair. If you wish to connect line segments in a plot such as this, the input file must provide the segments' common endpoints twice, once each before and after the line that changes `plt`'s line style.

Figure 6.2: The ten symbols that can be plotted using plotstyles `c` (suboptions 30 through 39), `E`, and `S`. Numeric (0-9) and mnemonic (“circle”, “square”, etc.) names appear to the right of each symbol; either form may be used when specifying a symbol to be used with plotstyles `E` or `S`. The open symbols (0 through 4) have opaque centers; these are particularly recommended for relatively dense scatter plots because it is still possible to distinguish and count individual data points even when the symbols partially overlap. The size of the symbols is determined by the current font’s point size (see the description of the `-sf` option in chapter 11), *Colors, Line Styles, and Fonts*, beginning on page 73.

`e+`c `e−`c `e:`c Takes three columns (x, y, and e). This sub-option makes a scatter plot, like `s` (see below), but including error bars. Each e value tells `plt` the size of the error associated with the corresponding x and y. When using the `e` sub-option, you may specify the character (default: "`*`") to use when plotting (x, y), and you must indicate if you wish to plot the top, bottom, or both halves of the error bars, as follows:

- `e+`c plot (x, y) as "c", with top half of error bars only
- `e−`c plot (x, y) as "c", with bottom half of error bars only
- `e:`c plot (x, y) as "c", with both top and bottom error bars; the ":" may be omitted unless c is either "+" or "−"

`E+`n `E−`n `E:`n Takes three columns (x, y, and e). Exactly like its counterpart, `e`, except that this sub-option allows you to plot a specified symbol (identified by a symbol number n between 0 and 9 inclusive) instead of a character.

`f` Takes three columns (x, y_0, and y_1) and fills the area defined by the normal plots of (x, y_0) and (x, y_1) in the current color (default: black). See figure 6.10, page 42.

`i` Takes two columns, x and y. The (x, y) pairs are plotted in impulse response form. See figure 6.11, page 43.

`l` Takes four columns, x_0, y_0, x_1, and y_1. The line segments with endpoints (x_0, y_0) and (x_1, y_1) are plotted. See figure 6.12, page 43.

`m` Takes one column (y). The x values are always taken from column 0 of the input. The output is a normal plot of (x, y) pairs. (If you supply only one data column number to `plt` and do not specify a plotstyle, this is the type of plot you will obtain by default.) See figure 6.13, page 44.

`n` Takes two columns, x and y. The output is a normal plot of (x, y) pairs. (If you supply two data column numbers to `plt` and do not specify a plotstyle, this is the type of plot you will obtain by default.) See figure 6.14, page 44.

`N` Takes two columns, x and y. The area bounded by the normal plot of the (x, y) pairs and the x axis is filled with the current color (default: black). See figure 6.15, page 45.

`o` Takes three columns (x, y_0, and y_1) and outlines the area defined by the normal plots of (x, y_0) and (x, y_1) in the current color (default: black). See figure 6.16, page 45.

`O` Takes three columns (x, y_0, and y_1), outlines the area defined by the normal plots of (x, y_0) and (x, y_1) in black, and fills this area in the current color (default: black). See figure 6.17, page 46.

`s`c Takes two columns, x and y. The output is a scatter plot of (x, y) pairs, where each pair is plotted using the character specified by c (a character suffixed to the plotstyle name, `s`). See figure 6.18, page 46.

`S`n Takes two columns, x and y. Exactly like its counterpart, `s`, except that this plotstyle allows you to plot a specified symbol (identified by a symbol number n between 0 and 9 inclusive), instead of a character. See figure 6.19, page 47.

`t` Takes three columns, x, y, and n. This plotstyle must be used together with the `-tf` or `-ts` option (see section 3.7). The n values specify strings from the `-tf` or `-ts` string array; `plt` prints these at the locations given by the corresponding (x, y) pairs (adjusting the string positions according to the text box coordinates provided to the `-tf` or `-ts` option). See figure 6.20, page 47.

6.1 A gallery of plotstyles

This section contains a series of plots made using many of the plotstyles described in the previous pages. Each plot was made using the same data file, `styles.data`, which contains:

```
0 0 0 0
1 -1 .25 1
2 -2 .5 2
3 1 .25 1
4 4 1 4
5 1 .25 1
6 -2 .5 2
7 -1 .25 1
8 0 0 0
```

In several of these examples, the data are plotted in 90% grey, to make clear how certain plotstyles use the plot color. `plt` is told to do this by the “`(G.90)`” appended to the plotstyle specification in each case; see chapter 11 for details on using color and grey level definitions such as these.

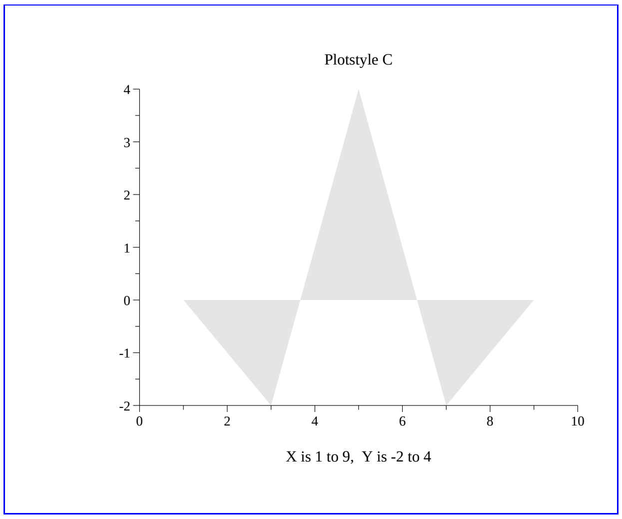


Figure 6.3: Produced using the command:

```
plt styles.data -p "0,1C(G.90)" -t "Plotstyle C"
```

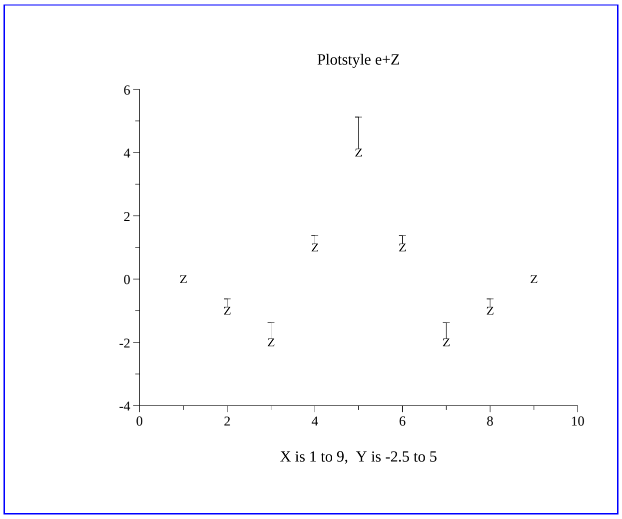


Figure 6.4: Produced using the command:

```
plt styles.data -p 0,1,2e+Z -t "Plotstyle e+Z"
```

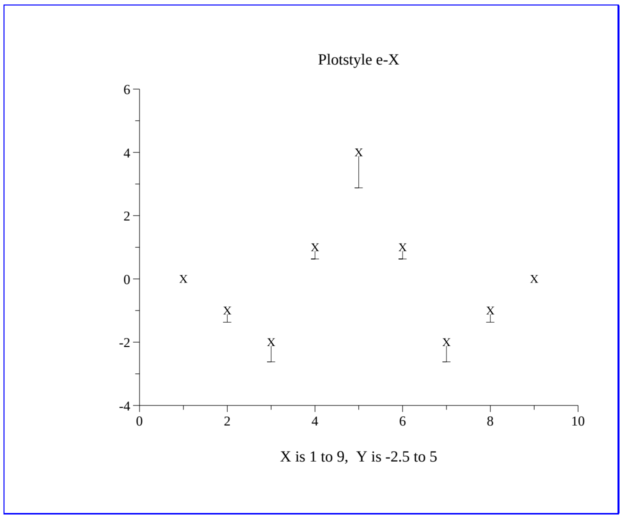


Figure 6.5: Produced using the command:

```
plt styles.data -p 0,1,2e-X -t "Plotstyle e-X"
```

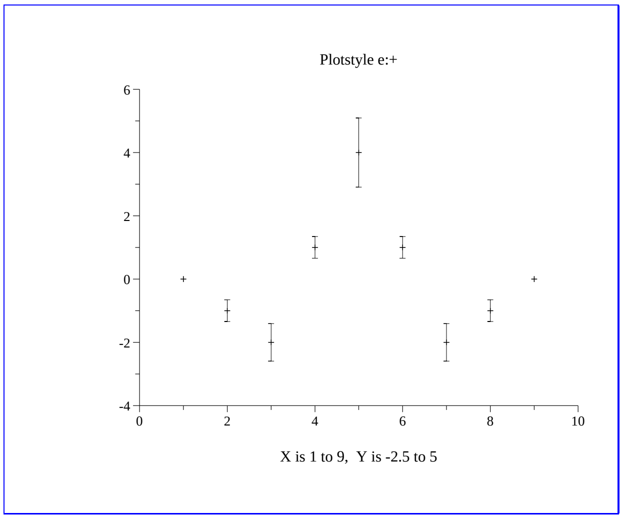


Figure 6.6: Produced using the command:

```
plt styles.data -p 0,1,2e:+ -t "Plotstyle e:+"
```

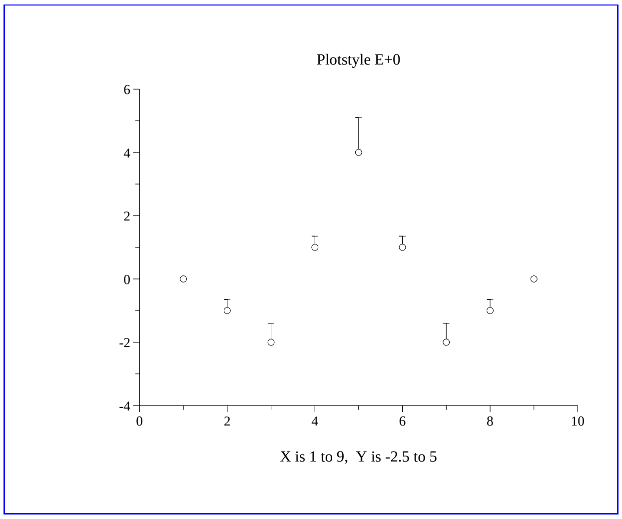


Figure 6.7: Produced using the command:

```
plt styles.data -p 0,1,2E+0 -t "Plotstyle E+0"
```

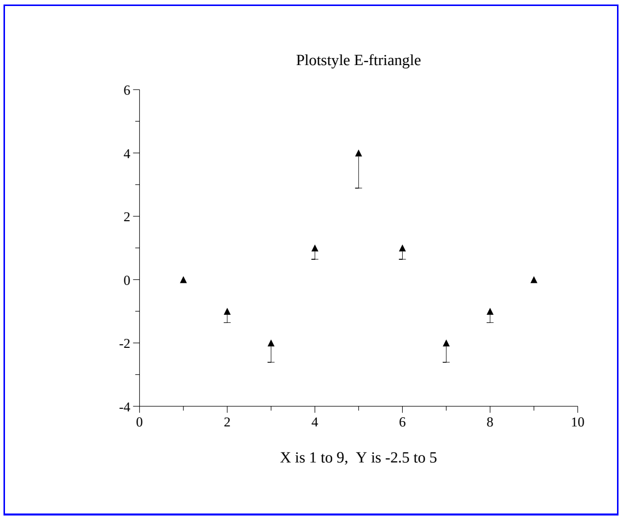


Figure 6.8: Produced using the command:

```
plt styles.data -p 0,1,2E-ftriangle -t "Plotstyle E-ftriangle"
```

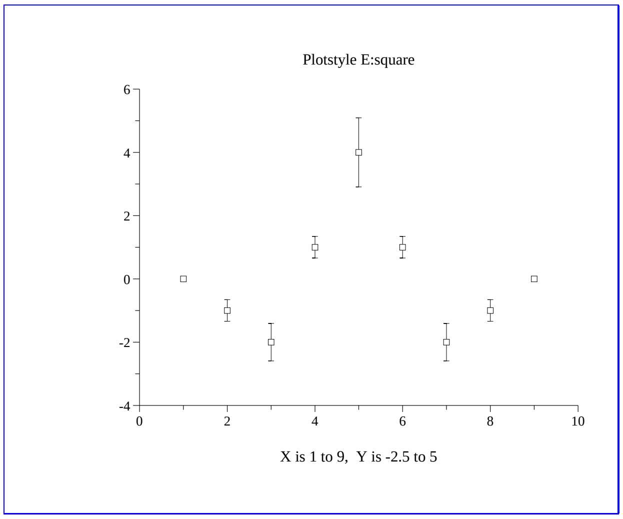


Figure 6.9: Produced using the command:

```
plt styles.data -p 0,1,2E:square -t "Plotstyle E:square"
```

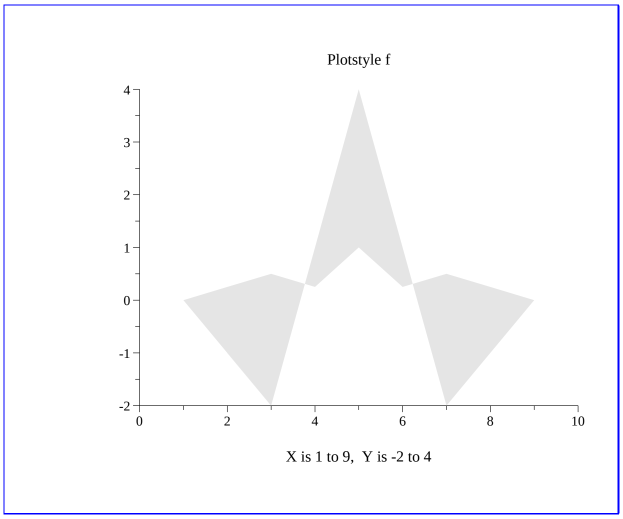


Figure 6.10: Produced using the command:

```
plt styles.data -p "0,1,2f(G.90)" -t "Plotstyle f"
```

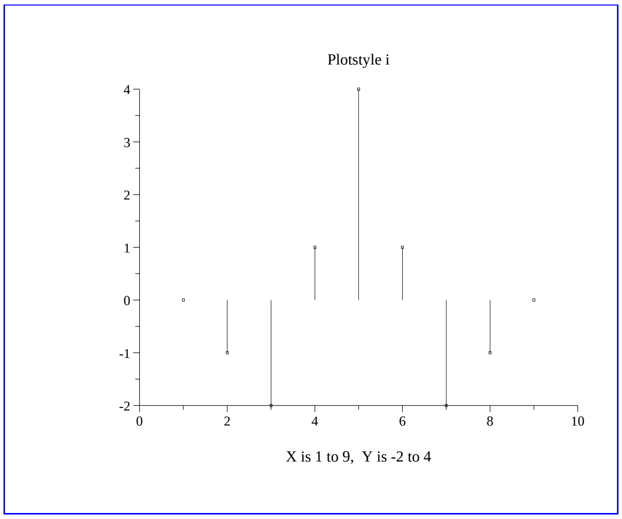


Figure 6.11: Produced using the command:

```
plt styles.data -p 0,1i -t "Plotstyle i"
```

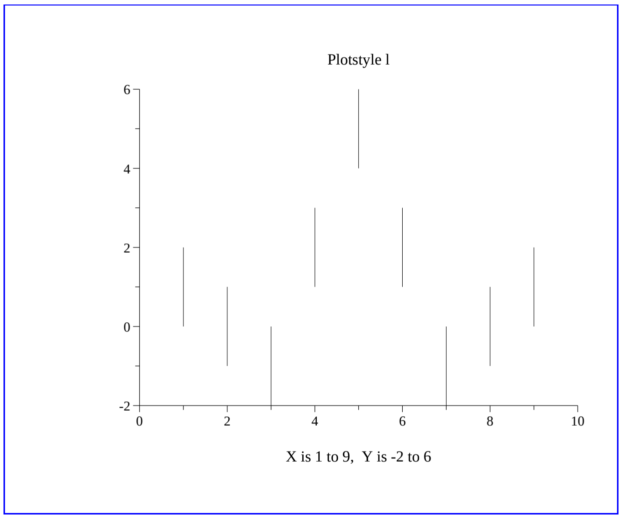


Figure 6.12: Produced using the command:

```
plt styles.data -p 0,1,0,3l -t "Plotstyle l"
```

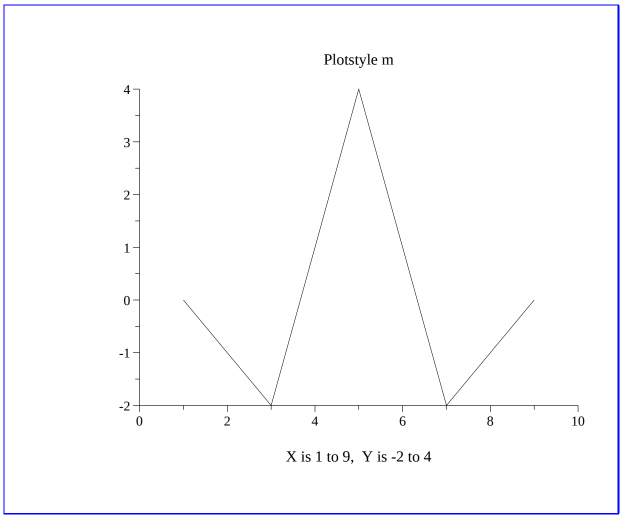


Figure 6.13: Produced using the command:

```
plt styles.data -p 1m -t "Plotstyle m"
```

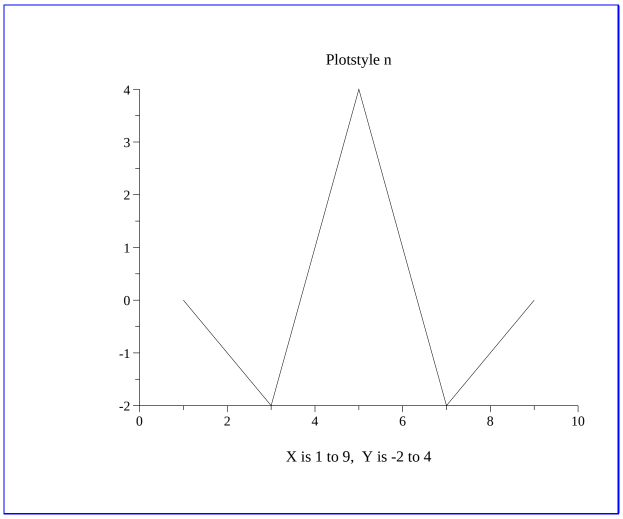


Figure 6.14: Produced using the command:

```
plt styles.data -p 0,1n -t "Plotstyle n"
```

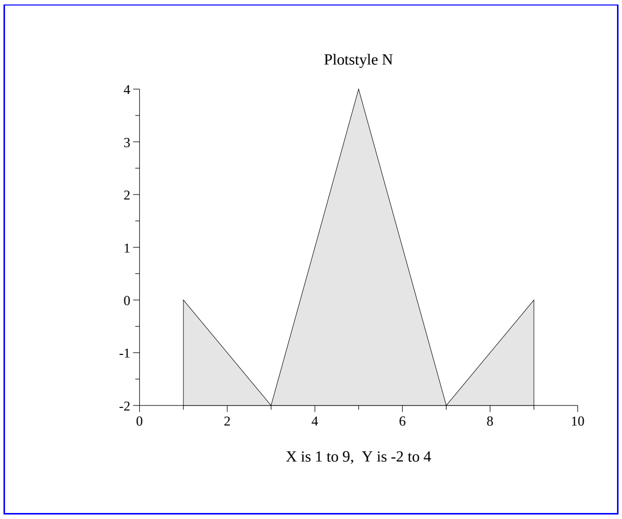


Figure 6.15: Produced using the command:

```
plt styles.data -p "0,1N(G.90)" -t "Plotstyle N"
```

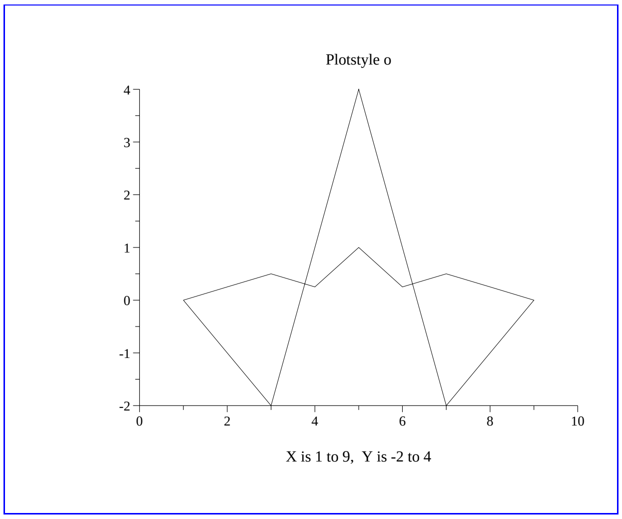


Figure 6.16: Produced using the command:

```
plt styles.data -p 0,1,2o -t "Plotstyle o"
```

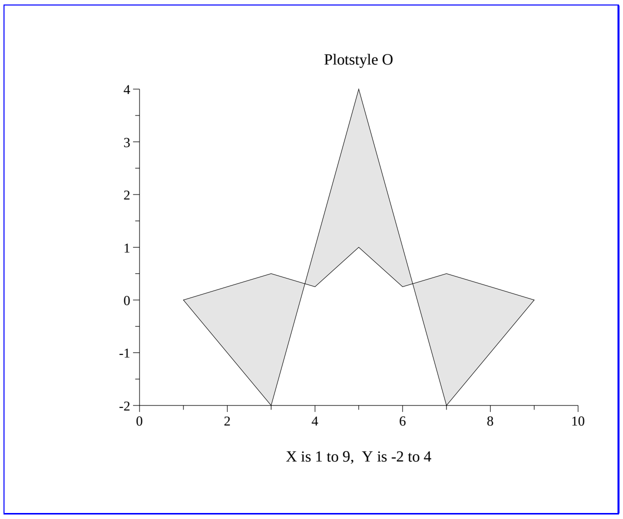


Figure 6.17: Produced using the command:

```
plt styles.data -p "0,1,2O(G.90)" -t "Plotstyle O"
```

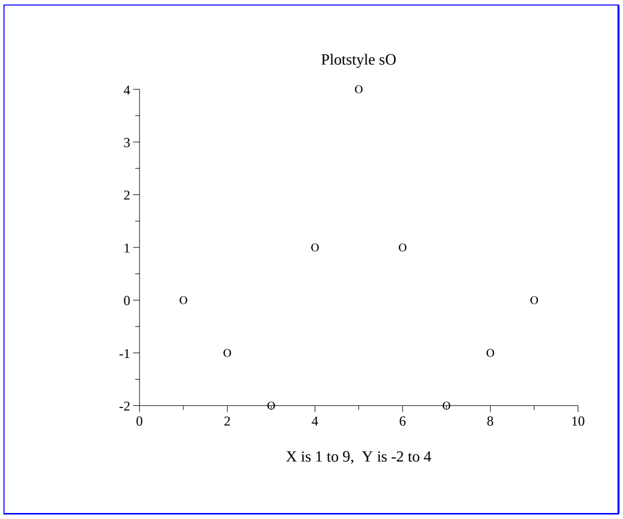


Figure 6.18: Produced using the command:

```
plt styles.data -p 0,1sO -t "Plotstyle sO"
```

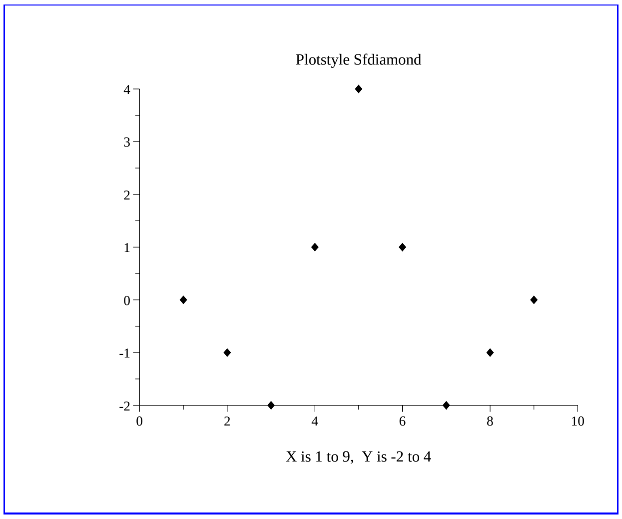


Figure 6.19: Produced using the command:

```
plt styles.data -p 0,1Sfdiamond -t "Plotstyle Sfdiamond"
```

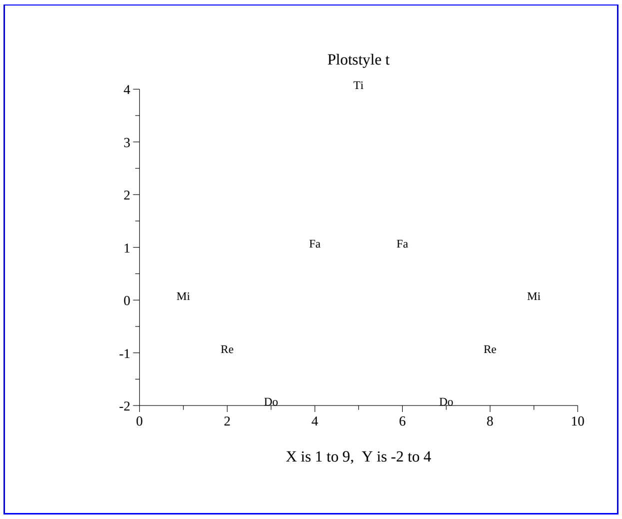


Figure 6.20: Produced using the command:

```
plt styles.data -p 0,1,3t -t "Plotstyle t" \
    -ts "Do Re Mi Fa Sol La Ti" CB
```

Chapter 7

Plotting Two or More Data Sets Together

Until now, most of the plots we have seen show only one data set. Often, as in figure 3.1, on page 23, it may be useful to plot two or more data sets together. There are several ways to arrange more than one plot on a page or window. This chapter shows first how you can overlay plots; how you can use different plot styles and create legends (or keys) so that overlaid plots can be distinguished; and finally how you can arrange plots side-by-side on a page or window.

7.1 Plotting multiple data sets on one set of axes

`plt` can plot more than one data set on the same set of axes. There are several ways to overlay plots.

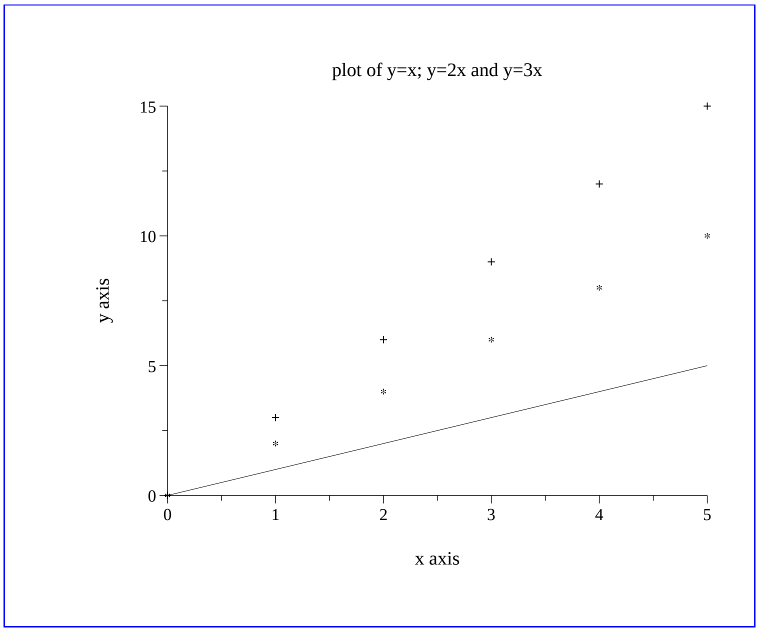
Figure 7.1 demonstrates the use of the `-p` option to specify multiple plots in a single invocation of `plt`, using a data file that includes a column of ordinates for each plot. The figure can be produced using any of these commands:

```
plt example5.data 0 3 0 2 1 -F"p s+ s* m"
    -x "x axis" -y "y axis" -t "plot of y=x; y=2x and y=3x"

plt example5.data 0 1 2 3 -F"p 0,3s+ 0,2s* 1m"
    -x "x axis" -y "y axis" -t "plot of y=x; y=2x and y=3x"

plt example5.data % -F"p 0,3s+ 0,2s* 1m"
    -x "x axis" -y "y axis" -t "plot of y=x; y=2x and y=3x"
```

There are several ways of specifying the data columns to be "taken" by the `-p` plotstyles. In the first command, the data columns are listed (after the name of the data file) in the order they are taken (the first '`s`' takes column 0 as x and column 3 as y, the second '`s`' takes column 0 as x and column 2 as y, and the '`m`' takes column 1 as y and



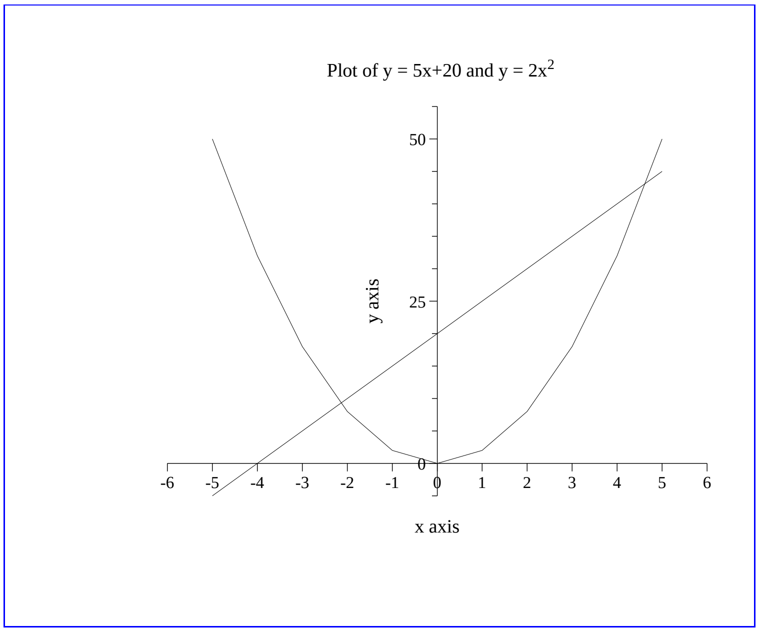
Data

0	0	0	0
1	1	2	3
2	2	4	6
3	3	6	9
4	4	8	12
5	5	10	15

Figure 7.1: Produced using the command:

```
plt example5.data 0 3 0 2 1 -F"p s+ s* m"
    -x "x axis" -y "y axis" -t "plot of y=x; y=2x and y=3x"
```

This command produces three traces on the same pair of axes (using "+" to make a scatter plot of columns 0 and 3, "*" to make another scatter plot of columns 0 and 2, and using columns 0 and 1 to make a normal plot).



Data

```
-5 50 -5
-4 32 0
-3 18 5
-2 8 10
-1 2 15
0 0 20
1 2 25
2 8 30
3 18 35
4 32 40
5 50 45
```

Figure 7.2: Overlaid plots (see text for explanation).

implicitly takes column 0 as x). In the second command, *all* of the data columns are listed in order after the name of the data file, and each plotstyle name is prefixed with a comma-separated list of the data columns it takes. The third command illustrates how "`%`" can be used as shorthand for "all of the data columns, in order"; it is equivalent to the second command.

If the data sets to be overlaid come from different files, it will be easiest to invoke `plt` once with each data file, and to use a different method to create the overlay. The `-fa` and `-o` options allow you to do this. Plot the data from the first file in the usual way, but use `-fa` to record the axis limits into a format file. Then plot the data from the second file using this format file and the `-o` option, which suppresses all output except your new data plot. Study figure 7.2, which overlays the plot of $y = 5x + 20$ over that of $y = 2x^2$. In this example, the axis specifications for the plot of $y = 2x^2$ are written into the format file `example7.axes`:

```
plt example7.data 0 1 -f example7.format
```

where `example7.format` contains:

```
t Plot of y = 5x+20 and y = 2x
L (P*.8) - - LB 2
x x axis
y y axis
xa -6 6 1 - 1 0
ya -5 55 5 - 5 0
fa example7.axes
```

[In `example7.format`, the second line (`L (P*.8) - - LB 2`) appends a small superscript 2 to the plot title. `(P*.8)` reduces the point size to 80% of the default, and "`- - LB"`" aligns the left bottom point of the 2 at the `RC` point of the previous string (the title). See chapter 11 for a discussion of how text styles, including point size, can be modified; and see chapter 8 for details on the `-L` option and on creating superscripts and subscripts in plot titles and other text.]

The information written by the command above into `example7.axes` is recalled in the command below (using the `-f` option), and the new plot is then overlaid using the `-o` option.

```
plt example7.data 0 2 -f example7.axes -o
```

Although a single data file, `example7.data`, contains data for both plots in this example, the data might just as easily have been collected from different files. Note that if you know ahead of time what axis limits you will use, you don't need `-fa`; just use `-xa`, `-ya`, `-X`, or `-Y` with the appropriate axis limits when making the second plot.

7.2 Legends

A legend (also known as a key) is useful when two or more data plots are drawn on the same axes, as in the previous section. A legend typically includes *sample plot segments* (short segments drawn using the same line styles used for the data plots), each of which is followed by *legend text* (typically a description of the associated plot). Figure 7.3 illustrates a plot that includes a boxed legend.

The `-lp` option is used to specify the location for the legend, and the `-le` option is used to define an entry to be shown as part of the legend. These options take their arguments as follows:

`-lp` *xw0 yw0 boxscale seglength opaque*
`-le` *linenumber plotnumber text*

The first two arguments of `-lp` define the window coordinates of the upper left corner (`LT`) of the legend text. `plt` draws a box around the legend text, determining the width and height of the box based on the size of the text and sample plot segments it encloses, but you can adjust the width of the box using the optional *boxscale* argument (which is a factor that multiplies the width calculated by `plt`). Thus a *boxscale* of 1.05 creates a box 5% larger than the default size; as a special case, a *boxscale* of 0 suppresses the box entirely. The optional *seglength* argument tells `plt` how long to

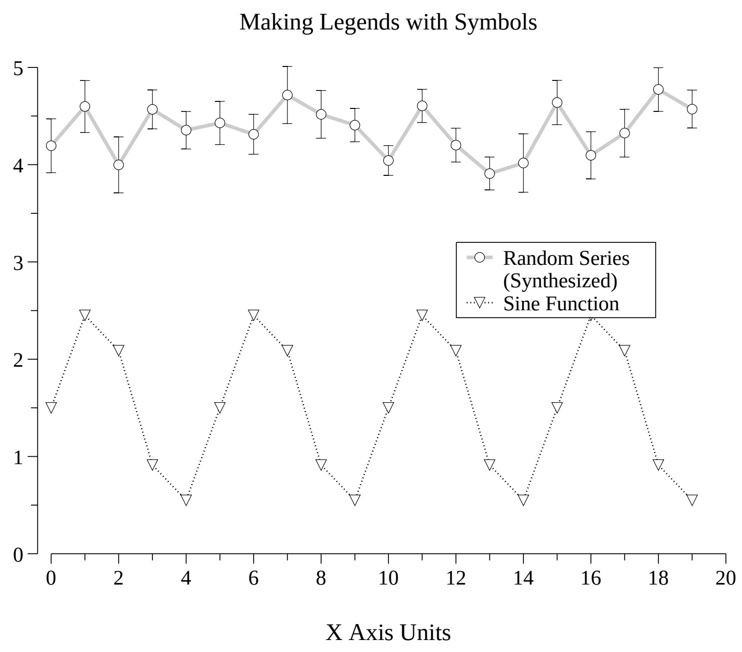


Figure 7.3: See section 7.2, *Legends*, for details on how this plot was created.

make the sample plot segments in the legend, as a fraction of the x axis length (by default, `plt` draws these segments so that they are 5% of the length of the x axis). If *opaque* is `yes` (default), the background of the legend box is opaque white; otherwise, the legend box is transparent (any previously drawn material remains visible through the legend box).

The first argument of `-le` specifies the line number of the entry within the legend (the top line is line number 0). Each plot drawn by `plt` has an implicit *plotnumber* (the first has a *plotnumber* of 0); this is the second argument of `-le`, used to determine the line style for the entry's sample plot segment). The third and final argument of `-le` is the text to be shown next to the sample plot segment.

Study figure 7.3 to see how these options work. This plot was produced using the command

```
plt example10.data % -f example10.format
```

where the data file, `example10.data`, was produced by this program fragment:

```
for (i = 0; i < 20; i++)
    printf("%d %.3lf %.3lf %.3lf\n",
           i,
           sin(0.4*i*PI) + 1.5,
           (double)random()/(double)RAND_MAX + 3.8,
           .1 + 0.15*(double)random()/(double)RAND_MAX);
```

and `example10.format` contains:

```
t Making Legends with Symbols
x X Axis Units
xa 0 20 1
ya 0 5

# Move the y axis slightly to the left.
yo .02

# Make four plots on the same set of axes.
p 0,1n(W5,L1) 0,1S1 0,2n(W12,G.8) 0,2,3E0

# Put legend text at window coordinates (.67,.64) and
# widen the legend box by 5%.
lp .67 .64 1.05

# Describe plot 2 in the top line (line 0) of the legend.
le 0 2 Random Series

# Plot 3 is the same as plot 2, but with a different
# plotstyle (using symbols).  By using a second "le"
# command for line 0 here, we create a sample plot
# segment that is composed by overlaying the two
# plotstyles.  The text has already been supplied in
# the previous command, so it is omitted here.
le 0 3

# The plot number is omitted from the next "le" command,
# because the text that is to appear on line 1 of the
# legend is a continuation of the description of the
# data in line 0.  The "-" means that no sample data
# segment will be drawn when this command is executed.
le 1 - (Synthesized)

# Describe plots 0 and 1.
le 2 0 Sine Function
le 2 1
```

(The `-yo` option is described in chapter 12.) The line beginning with “`p 0`...”

makes four plots using several different plot styles:

- Plot number 0 (`0,1n(W5,L1)`) plots columns 0 and 1 in normal plot (`n`) style, using a line width of 5 (`W5`, slightly broader than the default, 3 or 4) and a dotted line (`L1`).
- Plot number 1 (`0,1S1`) replots columns 0 and 1, this time as a scatter plot (`S1`), where the "`1`" indicates the use of symbol number 1 (an open inverted triangle).
- Plot number 2 (`0,2n(W12,G.8)`) plots columns 0 and 2 in normal plot style, using a broad line (`W12`) drawn in 80% grey (`G.8`).
- Plot number 3 (`0,2,3E0`) replots columns 0 and 2 as a scatter plot with error bars from column 3, using symbol number 0 (an open circle).

Transient fontgroup modifications such as "`(W5,L1)`" and "`(W12,G.8)`" are described in detail in chapter 11. Note that the order of the overlay affects the appearance; since the open symbols (numbers 0 through 4) used in plots 1 and 3 have opaque centers, they appear differently if drawn on top of (after) other elements than if drawn below (before) those elements.

7.3 Placing multiple plots on a page

In this section, the concept of using multiple windows within a single screen or page is introduced. You will need to become familiar with the following options:

`-W` *xp0 yp0 xp1 yp1*
`-w` *type subsection*

The arguments of the `-W` option are the *page coordinates* of the lower left $(xp0, yp0)$ and upper right $(xp1, yp1)$ corners of the plot window. (Recall that the default plot window, which fills the entire page or screen, has corners with page coordinates $(0, 0)$ and $(1, 1)$.) All output is scaled to fit within this window.

Study figure 7.4, in which the `-W` option is used to scale entire plots and the `-s e` option (see chapter 10) is used to suppress erasure so that the plots appear on the same page.

The `-w` option can address several windows that have been predefined for convenience. The first argument of `-w` may be `m` (a single window that fits within a frame to be described shortly), `b` (one of two windows covering either the top or bottom of the page), or `q` (one of four windows covering the upper left, upper right, lower left, or lower right quadrant of the page). The variant types `ms`, `bs`, and `qs` are windows of reduced size to allow extra room for axis labels. Use the *subsection* argument (which may have values 0, 1, 2, 3, or 4) to specify which window is to be used for the current plot. By convention, subsection 0 is used to start a new page (erase the screen) and to draw a grid and a title for the page. When using subsections 1, 2, 3, or 4, the plot is drawn on the current page without erasing anything that may have been plotted there already.

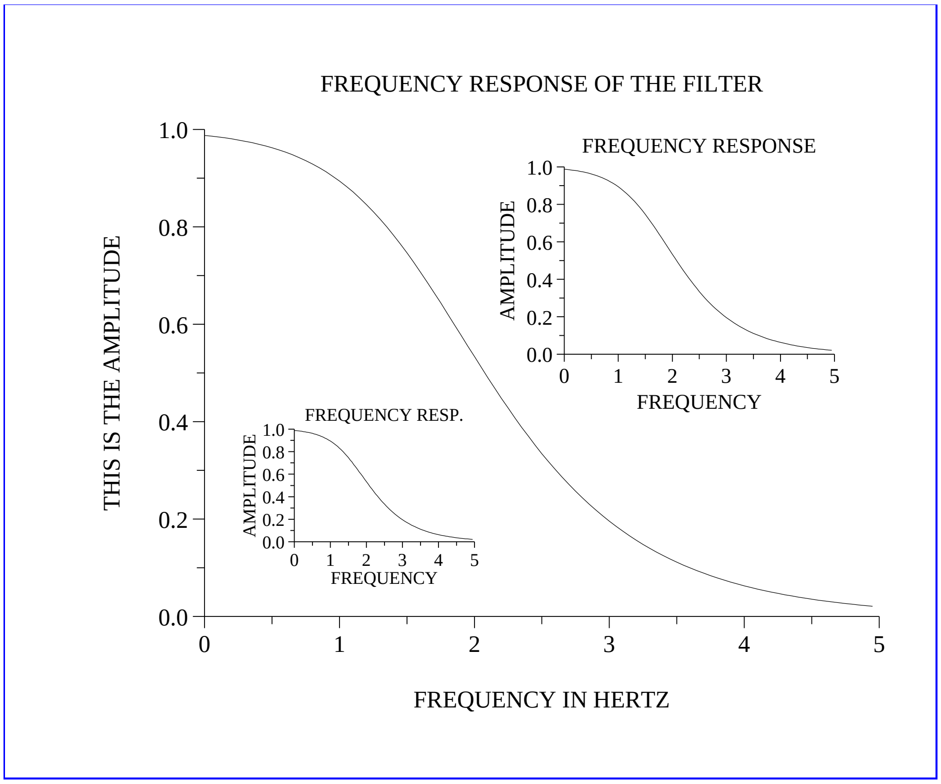


Figure 7.4: This example demonstrates how a plot can be scaled and placed on the page; it was produced using these commands:

```
plt ldemo.data 2 3 -t"FREQUENCY RESPONSE OF THE FILTER" \
 -x"FREQUENCY IN HERTZ" -y"THIS IS THE AMPLITUDE" -sf all P16
plt ldemo.data 2 3 -t"FREQUENCY RESP." -x"FREQUENCY" \
 -y"AMPLITUDE"  -W .3 .3 .5 .45 -se -sf all P12
plt ldemo.data 2 3 -t"FREQUENCY RESPONSE" -x"FREQUENCY" \
 -y"AMPLITUDE"  -W .6 .55 .9 .8 -se -sf all P14
```

Figures 7.5, 7.6, and 7.7 illustrate the use of `-wm`, `-wb`, and `-wq` respectively. All three use the data file `example11.data`, which contains:

```
0 0 0 9 1
1 3 1 6 0
2 2 2 7 1
3 4 3 4 2
4 3 4 5 3
5 5 3 3 4
6 4 2 4 3
7 7 1 2 2
8 6 0 3 1
9 9 1 0 0
```

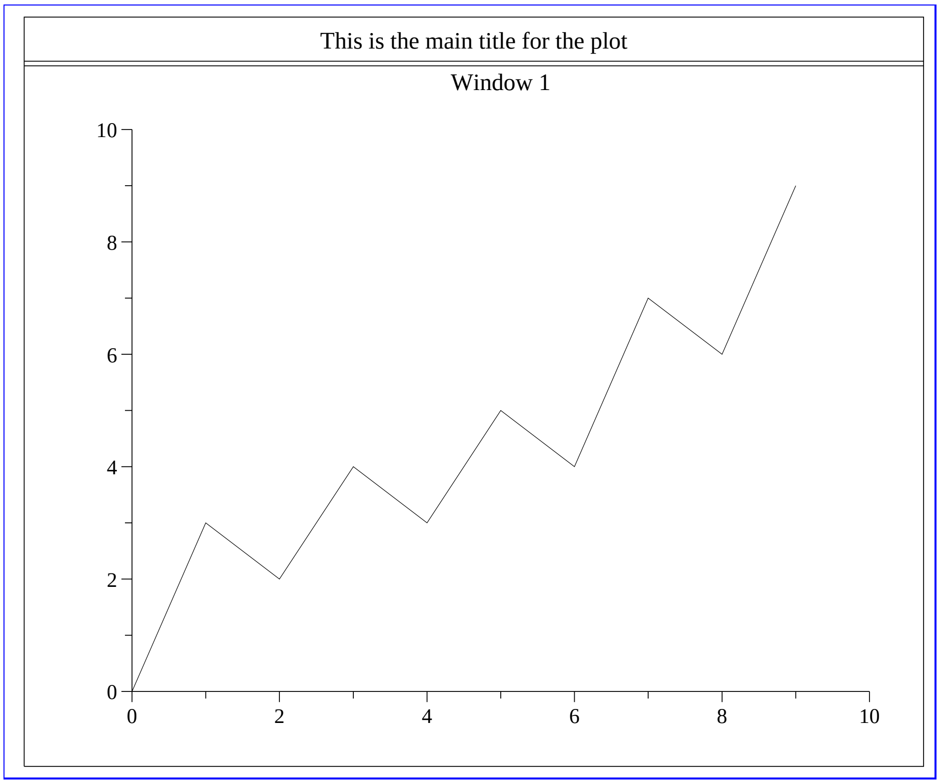


Figure 7.5: This plot was created using the commands:

```
plt -wm 0 -t "This is the main title for the plot"
plt example11.data 0 1 -wm 1 -t "Window 1"
```

When reducing the size of a plot, it is often desirable to avoid rescaling any text in the plot, so that it remains readable. The `-ch` option makes this possible:

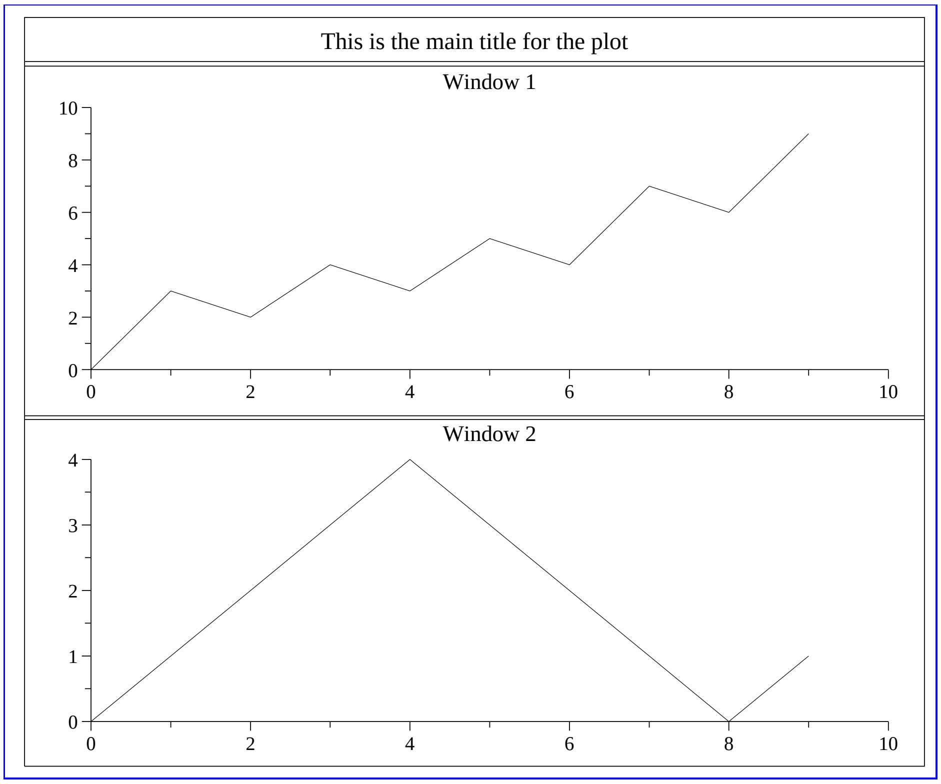


Figure 7.6: This plot was created using the commands:

```
plt -wb 0 -t "This is the main title for the plot"
plt example11.data 0 1 -wb 1 -t "Window 1"
plt example11.data 0 2 -wb 2 -t "Window 2"
```

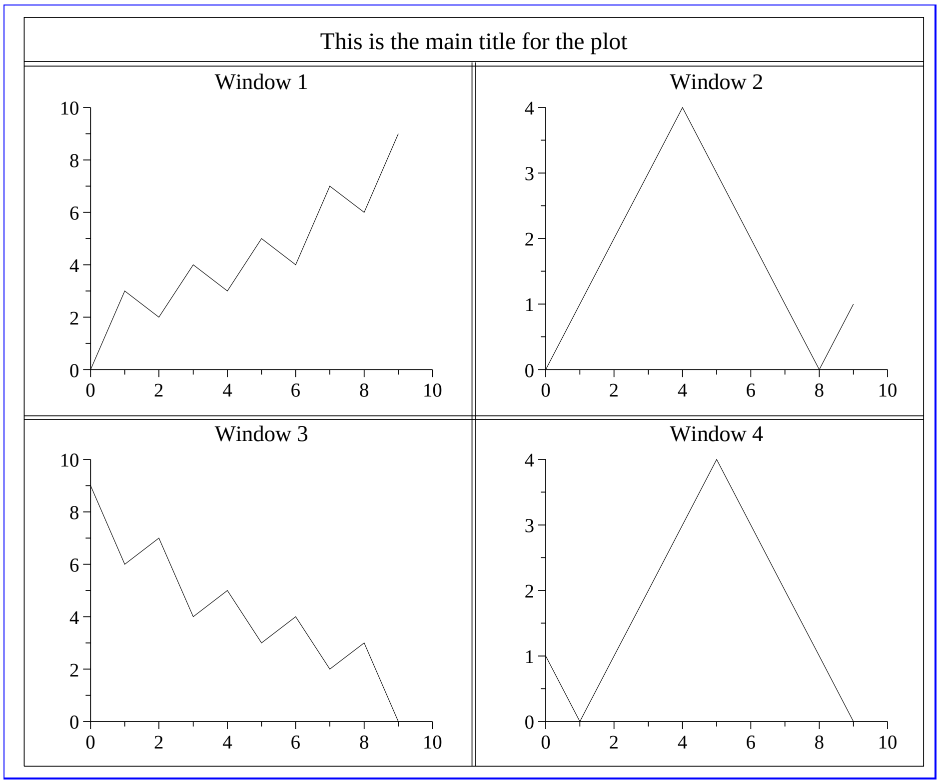


Figure 7.7: This plot was created using the commands:

```
plt -wq 0 -t "This is the main title for the plot"
plt example11.data 0 1 -wq 1 -t "Window 1"
plt example11.data 0 2 -wq 2 -t "Window 2"
plt example11.data 0 3 -wq 3 -t "Window 3"
plt example11.data 0 4 -wq 4 -t "Window 4"
```

`-ch` ***height-factor width-factor*** Use this option to resize text; the arguments specify factors by which the normal height and width of characters output by `plt` are to be enlarged (or reduced, if a factor is less than 1).

Chapter 8

Labelling Your Plot

The `-hl` and `-vl` options allow you to create horizontal and vertical labels anywhere within your plot, using window coordinates, (xw, yw), to specify the location for the label; and a text box coordinate, *tbc* (e.g., "`CB`", "`RT`", etc.), to specify which point in the text box should coincide with (xw, yw). These options take their arguments as follows:

`-hl` *xw yw tbc nlines file*
`-vl` *xw yw tbc nlines file*

All arguments are optional (but if a specified argument follows an omitted argument, you must supply a "`-`" as a placeholder for the omitted argument). If xw or yw is omitted, the missing coordinate is set to that of the `RC` point of the previous label, allowing labels to be concatenated (see below). If *tbc* is omitted, the default value is "`CC`", i.e., the label will be centered on (xw, yw). If specified, *nlines* is a number indicating how many input lines are to be used to form the label (line breaks are reproduced by `plt` as they appear in the input).

If *file* is specified, the text for the label is read from the first *nlines* lines of *file*. If *nlines* is not specified, the entire contents of *file* are used for the label.

If *file* is not specified, the next *nlines* lines of the `plt -f` format file or `plt -F` format string form the label; in this case, if *nlines* is also not specified, the label is formed from only one line (the next line) of the format file or string.

The `-L` and `-l` options can be used in a similar way to create horizontal labels. The arguments for these options are supplied as follows:

`-L` *xw yw tbc text*
`-l` *x y tbc text*

As for `-hl`, the *tbc* argument is one of the twelve named text box coordinates. The *text* argument is the string to be used for the label; it may include whitespace, but it ends at the end of the line on which it appears. The difference between `-L` and `-l` is that the former takes window coordinates (xw, yw), and the latter takes data coordinates (x, y).

These options are demonstrated in figure 8.1, which uses the format file `example9.format`, containing:

```
xa 0 600
ya 0 500
x Time Between Stimulations (msecs)
y Refractory Period (msecs)
# Note: the next line ends with a lower-case L, not a numeral 1!
p 0,1,2,3l
t ch 1.2
l 400 200 CC fires every beat
L .1 .9 CC . . .
hl .45 .81 CC 2
fires every
other beat
hl .25 .92 - 4
  fires
 every
third
beat
```

and the data file `example9.data`, containing:

```
0 0 500 500
0 0 250 500
0 0 166 500
0 0 125 500
0 0 100 500
```

In figure 8.1, notice especially how whitespace at the beginning of two lines in the second `hl` option's text is preserved in the plot. Note also how both data and window coordinates are used with several `plt` options. The next chapter describes several more options that can be used in the same way. Options `-A`, `-B`, `-C`, `-D`, `-L`, `-hl`, `-vl`, `-le`, and `-lp` accept window coordinates; options `-a`, `-b`, `-c`, `-d`, and `-l` accept data coordinates. For now, note that you often have a choice of using data or window coordinates, since many of these options come in two varieties: upper-case options that use window coordinates, and similar lower-case options that use data coordinates.

8.1 Concatenating labels

Labels can be concatenated if you wish to incorporate font changes, subscripts or superscripts, or other special effects in labels. The plot title in figure 7.2, page 51, illustrates how this can be done. `plt` keeps track of the `RC` point of each label, and uses these coordinates in place of omitted (x, y) or (xw, yw) coordinates in label options. If the coordinates are omitted from the first label in any given plot, the `RC` point of the plot title is used to determine the label coordinates.

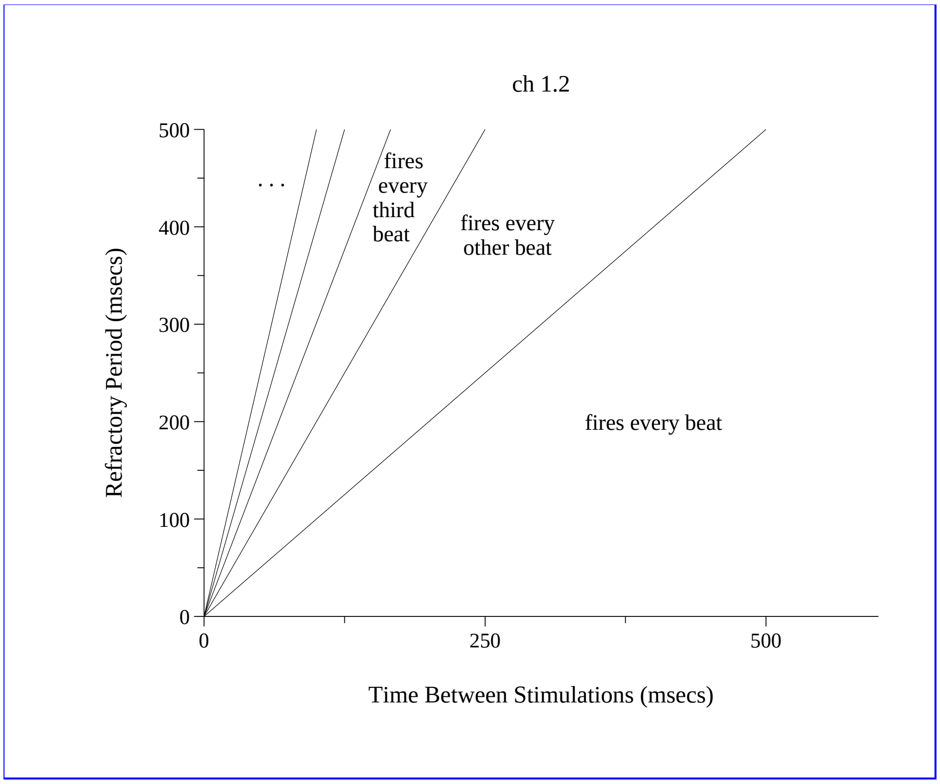


Figure 8.1: Labelling a plot using the `-hl`, `-l`, and `-L` options. This plot was produced using the command

```
plt example9.data % -f example9.format
```

This facility would be of little use if it were not possible to change the style used by `plt` for rendering the text of labels. This can be done using *fontgroup substitution*, as described in detail in chapter 11, beginning on page 11. In a nutshell, what you do is to describe the style you want in an extra argument that is enclosed in parentheses and that immediately follows the option name, as in:

```
-L (Ft-b-i,P20,Cmaroon) .5 .5 CC Marooned!
```

(This sets the label "Marooned!" in 20-point Times Bold Italic of an appropriate color.)

If you wish simply to concatenate labels, perhaps changing the font or text color, use `LC` as the text box coordinate for the second (and subsequent) labels, as in:

```
-L (Cblue,Fh) .4 .7 CC Hello,
-L (Cgreen,Fh-o) - - LC " World!"
```

These commands produce a blue Helvetica "Hello," centered on window coordinates (.4,.7) followed by a green Helvetica-Oblique "World!". Note that the entire string is not centered on (.4,.7); if you want to center a multi-part label you will need to do some experimentation. Also note the use of quotation marks to force the inclusion of the initial space before "World!".

In the following example, Greek and Roman letters are mixed to produce the equation $\mathrm{y} = \mathrm{A}\sin(\omega\mathrm{t} + \phi)$:

```
-L .3 .5 CC y = A sin(
-L (Fs) - - LC w
-L - - LC "t + "
-L (Fs) - - LC f
-L - - LC )
```

To create a subscript or subscript, use `LN` or `LB` respectively as the text box coordinate for the label containing the subscript or superscript, and use `LB` or `LN` to return to the original baseline afterwards. It is often effective to reduce the point size of superscripts and subscripts, as in this example, which produces $y = x^\alpha + x_0\beta$:

```
L .6 .3 LC y = x
L (Fs,P*.8) - - LB a
L - - LN " + x"
L (P*.8) - - LN 0
L (Fs) - - LB b
```

The same effects are possible in a vertical label, as in this example:

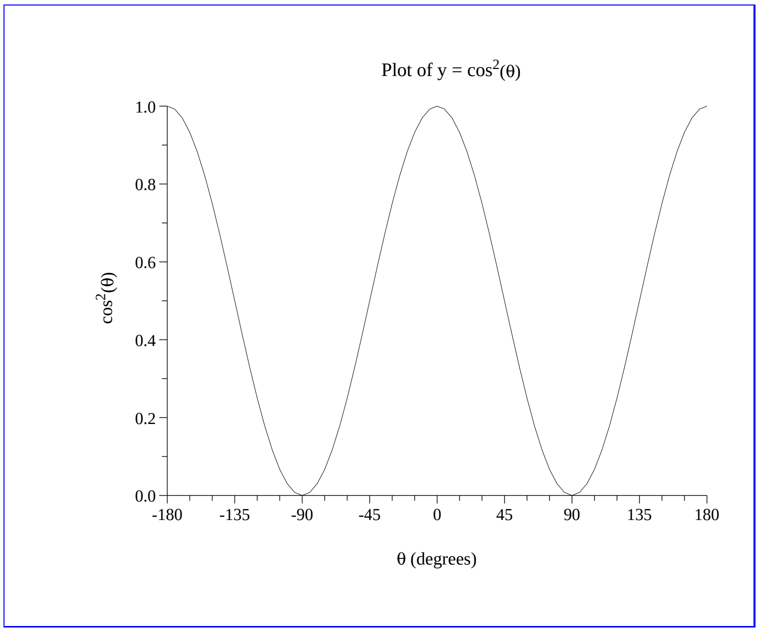


Figure 8.2: Labels with superscripts and font changes

```
vl .6 .5 LC 1 -
y = x
vl (Fs,P*.8) - - LB 1 -
a
vl - - LN 1 -
 + x
vl (P*.8) - - LN 1 -
0
vl (Fs) - - LB 1 -
b
```

See figure 8.2 for examples of this technique. This figure was created using the command

```
plt cos2.data 0 1 -f labels.format
```

where `cos2.data` was created using this program fragment:

```
for (i = -180; i <= 180; i += 5) {
    c = cos(M_PI*i/180.0);
    printf("%d %g\n", i, c*c);
}
```

and `labels.format` contains:

```
xa -180 180 15 - 3
ya 0 1

# Set the first part of the plot title using the -t option.
t Plot of y = cos
# Concatenate the remaining pieces using -L options.
L (P*.8) - - LB 2
L - - LN (
L (Fs) - - LC q
# "q" in Symbol font is theta.
L - - LC )

# Avoid drawing a normal x-axis label.
x ""
# Instead of a normal x-axis label, draw this label using
# the -L option with ym = -.16, to put it in the same
# position as a normal x-axis label.
L (Fs) .425 -.16 LC q
L - - LC " (degrees)"

# Similarly, draw this label using -vl with xm = -.115 to put it
# in the same position as a normal y-axis label.  (We don't need
# to suppress the normal y-axis label, because plt produces one
# only if requested explicitly.)
vl -.115 .44 LC 1 -
cos
vl (P*.8) - - LB 1 -
2
vl - - LN 1 -
(
vl (Fs) - - LC 1 -
q
vl - - LC 1 -
)
```

Another way to create subscripts and superscripts is to specify an appropriate y or yw while omitting the x coordinate (for horizontal labels). Some experimentation may be needed to achieve the desired results.

These methods cannot be used directly to concatenate text in axis labels. If you need to do this, suppress `plt`'s standard axis label, either by supplying an empty string as the argument for `-x` or `-y`, or by using the appropriate `-s` option (see chapter 10); then create your own axis label using the options described above.

Chapter 9

Drawing Line Segments, Arrows, and Boxes

Well-chosen annotations can make a good plot better. In the previous chapter, we saw how labels can be placed anywhere on a plot, using either data or window coordinates. `plt` also allows you to draw arbitrary line segments, arrows, and boxes on your plot.

Use the `-C` and `-c` (connect) options to draw arbitrary line segments, as follows:

> `-C` *xw0 yw0 xw1 yw1*
> `-c` *x0 y0 x1 y1*

These options connect the two points specified by their arguments with a line segment. The `-C` option connects the points with window coordinates $(xw0, yw0)$ and $(xw1, yw1)$. The `-c` option connects the points with data coordinates $(x0, y0)$ and $(x1, y1)$.

The `-A` and `-a` (arrow) options work in the same way as `-C` and `-c`, respectively, but they also draw an arrowhead pointing toward the first specified point, $(xw0, yw0)$ or $(x0, y0)$:

> `-A` *xw0 yw0 xw1 yw1*
> `-a` *x0 y0 x1 y1*

Note that the size of the arrowhead is determined by the specified point size in the figure (`f`) font group, or in local override instructions (see figure 11.4).

The `-B` and `-b` (box) options similarly accept arguments specifying two points, but in this case the points are diagonally opposite corners of a box (rectangle) that is to be drawn:

> `-B` *xw0 yw0 xw1 yw1*
> `-b` *x0 y0 x1 y1*

The `-D` and `-d` (dark box) options work in the same way as `-B` and `-b` respectively, but boxes drawn when using these options are filled. The filling is black by default, but

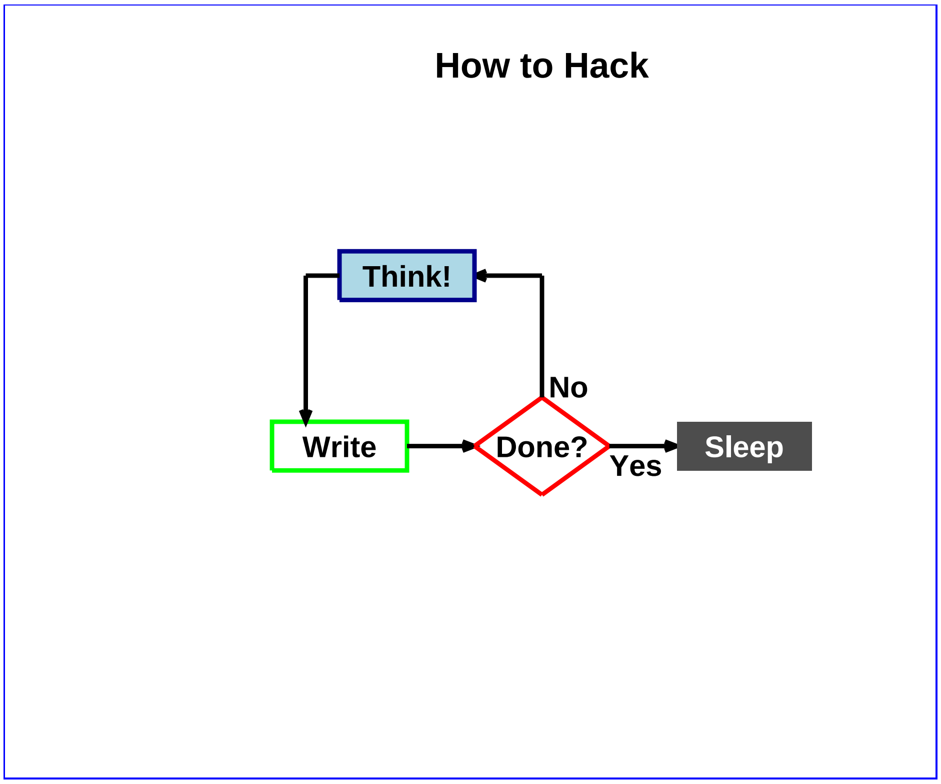


Figure 9.1: Line segments, arrows, and boxes

may be changed by setting the grey level or color in the `f` (figure) font group (see figure 11.5).

`-D` *xw0 yw0 xw1 yw1*
`-d` *x0 y0 x1 y1*

Remember that in each case, as with `-L` and `-l`, the upper case version of the option takes window coordinates, and the lower case version takes data coordinates.

These options are illustrated in figure 9.1, which was produced using the command:

```
plt -f flowchart.format
```

where the format file, `flowchart.format`, contains:

```
# Define axis ranges, but don’t draw the axes.
X 0 1
Y 0 1
s xy
# Use black, wide lines for figures by default.
sf f Cblack,W15,P20
# Draw labels in black 20 point Helvetica Bold.
sf l Cblack,P20,Fh-b
# Draw the plot title in 24 point Helvetica Bold.
t (Fh-b,P24) How to Hack
# Draw a green-outlined box and label it.
B (Cgreen) .1 .3 .3 .4
L .2 .35 CC Write
# Draw an arrow from the right edge of the box.
A .4 .35 .3 .35
# Draw a diamond using line segments, and label it.
C (Cred) .4 .35 .5 .45
C (Cred) .5 .45 .6 .35
C (Cred) .6 .35 .5 .25
C (Cred) .5 .25 .4 .35
L .5 .35 CC Done?
# Draw a line and arrow from the top of the diamond.
C .5 .45 .5 .7
A .4 .7 .5 .7
# Label the line and arrow.
L .51 .45 LB No
# Draw a filled light blue box.
D (CLightBlue) .2 .65 .4 .75
# Outline it in dark blue, and label it.
B (CDarkBlue) .2 .65 .4 .75
L .3 .7 CC Think!
# Draw a line and arrow from this box to the first.
C .2 .7 .15 .7
A .15 .4 .15 .7
# Draw an arrow from the right corner of the diamond.
A .7 .35 .6 .35
# Label the arrow.
L .6 .35 LT Yes
# Draw a filled box in 30% grey, and label it in white.
D (Cgrey30) .7 .3 .9 .4
L (Cwhite) .8 .35 CC Sleep
```

The `-s` option, used in this example to suppress the axes, is described in the next chapter. The `-sf` option, used to set the default characteristics for the figures and labels in the example, is discussed in chapter 11, beginning on page 73.

Chapter 10

Suppressing Plot Elements

The `-s` option allows you to suppress (i.e., to avoid drawing) selected features of your plots. The sub-options following `-s` specify which elements to suppress. The sub-options are as follows:

`e`	don't begin a new page (erase screen) before plotting
`a`	suppress anything associated with axes
`x`	suppress anything associated with the x axis
`y`	suppress anything associated with the y axis
`g`	suppress the grid
`m`	suppress x axis and y axis tick marks
`n`	suppress x axis and y axis tick mark numbers
`t`	suppress titles for x axis, y axis, and graph
`l`	suppress user-supplied labels (`-hl`, `-vl`, `-l`, and `-L` options)
`p`	suppress all data plotting
`f`	suppress all "figures" (boxes, line segments, arrows, and legends)
`C`	enable all plot features (undo effects of previous `-s` sub-options)
`X`	subsequent `-s` sub-options apply only to the x axis
`Y`	subsequent `-s` sub-options apply only to the y axis
`A`	subsequent `-s` sub-options apply to both axes (default)

Two or more sub-options can be grouped, as in:

```
-s eg
```

By default, data points that fall outside of the axis limits are suppressed. If any data points are not plotted for this reason, `plt` produces a warning message indicating the number of points that were excluded from the plot. The `-ex` option overrides this behavior:

`-ex` Use this option to ensure that all data points that fall on the page will be plotted.

Chapter 11

Colors, Line Styles, and Fonts

The user can separately control how text and line segments are rendered in five different contexts: axes and their numbering (`a`), figures (lines, boxes, and arrows, `f`), labels (`l`), data plots (`p`), axis and graph titles (`t`), and all of these at once (`all`). The contexts (`a`, `f`, `l`, `p`, and `t`) are the five *predefined font groups*. The table below matches each of the predefined font groups with the `plt` options having related outputs.

Font group	*Options*
`a`	`xa`, `ya`, `X`, `Y` (axes and numbering)
`f`	`A`, `a`, `B`, `b`, `C`, `c`, `D`, `d`, `le`, `lp` (figures and legends)
`l`	`L`, `l`, `hl`, `vl` (labels)
`p`	`p` and sub-options (data plots)
`t`	`x`, `y`, `t` (titles)

A font group is defined by the following six parameters:

`F` font name (see figure 11.1), one of:

`c`	courier	`h`	helvetica	`t`	times
`c-b`	courier-bold	`h-b`	helvetica-bold	`t-b`	times-bold
`c-o`	courier-oblique	`h-o`	helvetica-oblique	`t-i`	times-italic
`c-b-o`	courier-bold-oblique	`h-b-o`	helvetica-bold-oblique	`t-b-i`	times-bold-italic
`s`	symbol				

`P` point size (an integer ≥ 0; 12 or 14 are normal point sizes)

`C` color (see appendix A, *Color Names*, beginning on page 91)

`G` grey level (see figure 11.2); the grey level can be between 0.0 (black) and 1.0 (white)

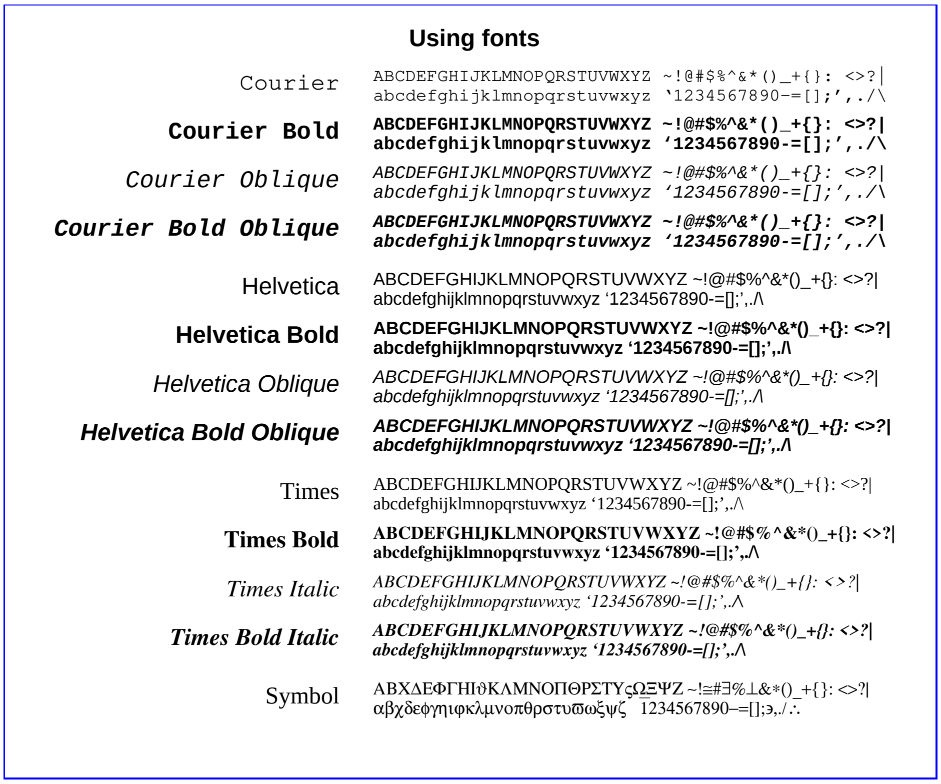


Figure 11.1: Samples of available fonts

`W` line width (an integer ≥ 0); a normal line width is 3 or 4. By convention, a line width of 0 produces the narrowest possible line; on some high-resolution devices, such lines may not be visible, however. See figure 11.2.

`L` line style (see figure 11.2); select from the following using either the number (0-4) or the one-word name:

0	`solid`
1	`dotted`
2	`shortdashed`
3	`dotdashed`
4	`longdashed`

Use the `-sf` option to define a font group or to modify the parameters of a predefined font group:

`-sf` *fontgroup specs ...*

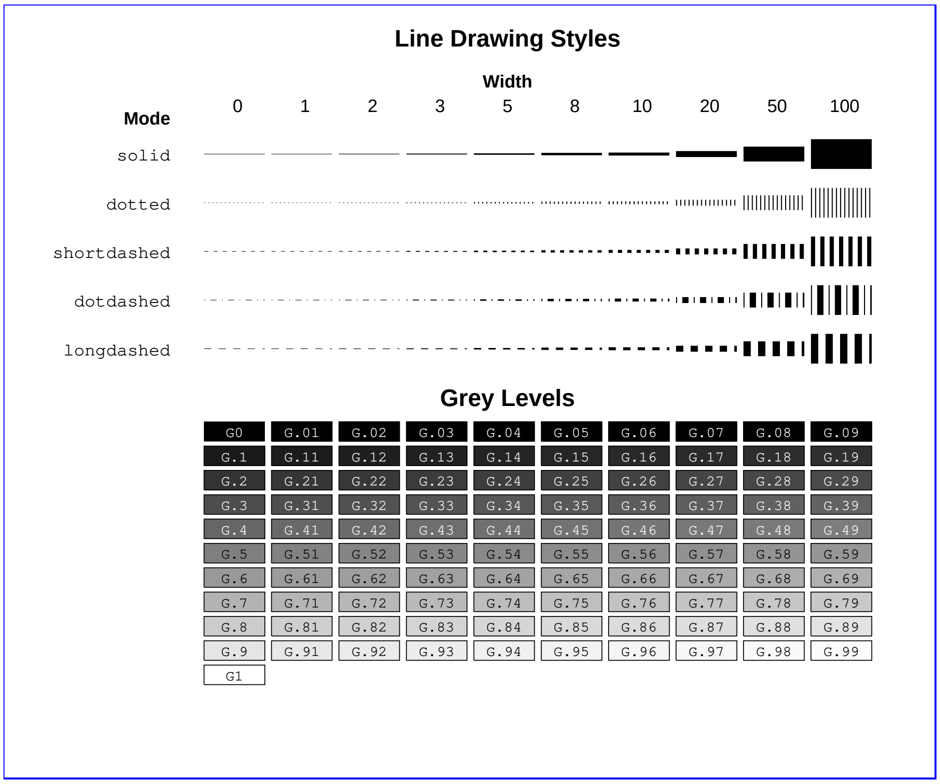


Figure 11.2: Line widths and styles, and grey levels. On very high resolution output devices, lines rendered with zero width `(W0)` may not be visible; a safer choice for a thin line is `(W1)`. On older PostScript printers, however, zero-width lines are usually visible because resolution is not too high, and they often are rendered much faster than lines of any other width. Note that grey levels can also be specified using colors; for example, the font group specification `(Cgrey45)` is equivalent to `(G.45)`.

The *fontgroup* argument is the name of the font group; this can be one of `a`, `f`, `l`, `p`, or `t`, or you can specify any other name you choose for a custom font group. The *specs* arguments can include any of the parameter names `F`, `P`, `C`, `G`, `W`, or `L`, followed immediately by the value to be given to the parameter; separate parameters with commas. For example, the following specifies that titles should be shown using 18-point Times Bold Oblique:

```
-sf t Ft-b-o,P18
```

Before it produces any output, `plt` scans the entire command line (and the format file, if `-f` is used) for font group definitions made using `-sf`. Thus the position of `-sf` among the other options makes no difference to the final result.

Fontgroup substitution can be used to make `plt` use any desired fontgroup to control the appearance of any desired element. Although each `plt` option that renders text or graphics has an associated font group that normally determines its appearance, any other font group can take on this function if you substitute it by naming it when using the option. For most `plt` options, do this by supplying the font group's name in parentheses immediately after the option, as in these examples:

```
-B "(p)" .1 .1 .3 .5
-L "(t)" .6 .4 CB "Group A (n=25)"
```

(The quotation marks should be omitted within a format file; they are needed on the command line to protect the parentheses from the shell.) When using fontgroup substitution after a `-p` option, however, the font group name should follow the plotstyle specification, as in:

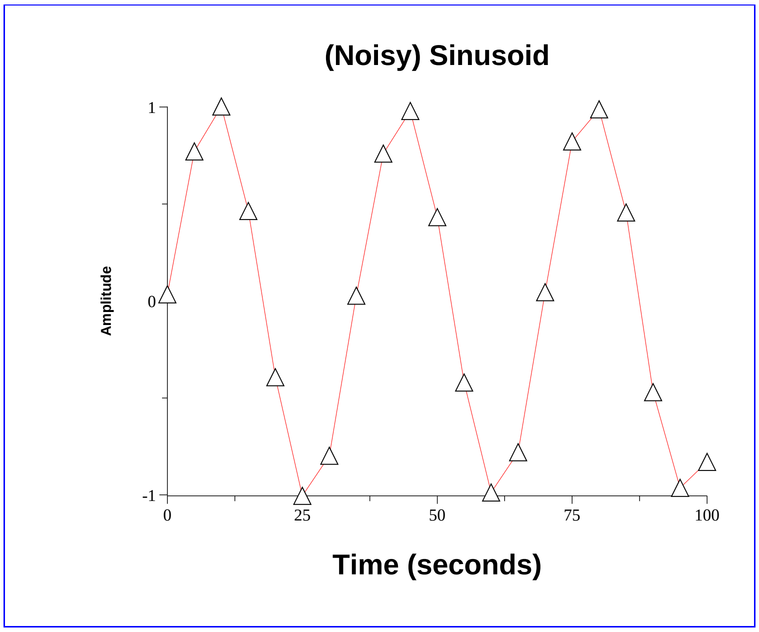
```
-p "0,1Striangle(t)"
```

Even more flexibility is possible using *transient fontgroup modifications*. These can be performed using the same syntax as for fontgroup substitutions, except that what goes inside the parentheses is a list of fontgroup parameters, for example:

```
-D "(Cmagenta)" .1 .6 .2 .7
-p "0,1n(W3,Ldashed)"
```

Transient fontgroup modifications apply only to the option in which they are defined.

In addition to absolute parameter values, as above, *relative parameter values* may be supplied in a font group specification. Relative parameter values are given in the form $<op>$ N, where *op* is one of the arithmetic operators (`+`, `-`, `*`, or `/`) and N is the amount by which the previous value of the parameter is to be increased, decreased, multiplied, or divided respectively. For example, `W*2` doubles the previous line width.



Data

```
0 0.034
5 0.771
10 1.003
15 0.464
20 -0.393
25 -1.005
30 -0.798
35 0.027
40 0.760
45 0.980
50 0.432
55 -0.421
60 -0.988
65 -0.780
70 0.045
75 0.823
80 0.988
85 0.456
90 -0.470
95 -0.964
100 -0.830
```

Figure 11.3: This example illustrates the effects of fontgroup redefinitions and substitutions, transient fontgroup modifications, and the use of relative parameter values in fontgroup specifications. See the main text for details.

Figure 11.3 was created using:

```
plt example14.data -f fontgroup.format
```

The data file, `example14.data`, contains 21 samples of a noisy sinusoid. The format file, `fontgroup.format`, contains:

```
# The entire format file is scanned for "sf" options before
# anything is rendered, so the x axis title is rendered in
# the font specified by the "sf t" option below.
x Time (seconds)

# Specify black 24-point Helvetica Bold for the "t" fontgroup.
# Although black is the default color, it needs to be specified
# explicitly (see below).
sf t Fh-b,P24,Cblack

# Specify red for the "p" fontgroup.
sf p Cred

# Here, a transient fontgroup modification (P/2) overrides the
# previously defined point size setting for the "t" font, so
# that the y axis title is rendered half as large as the
# (unscaled) x axis title.
y (P/2) Amplitude

# As shown in the previous line, if the argument following
# an option begins with "(", it is interpreted as a local
# fontgroup spec.  In order to use a string beginning with
# "(" as a string argument, it is necessary to quote it:
t "(Noisy) Sinusoid"
# Note that this title is rendered in 24-point type;  the
# effect of the local spec used earlier is limited to the
# line on which it appeared.

# Here, we make two plots, first a normal plot using the
# "p" fontgroup (in red), then a scatter plot using triangles.
# The effect of the "(t)" is to specify that the "t" fontgroup
# must be used for the second plot;  thus the triangles are
# black, not red, and are sized to match the 24-point type of
# fontgroup "t".
p 0,1n(p) 0,1Striangle(t)
# If, however, the color had not been specified explicitly
# for fontgroup "t", plt would not change the color between
# plots, and the triangles would be rendered in red.
```

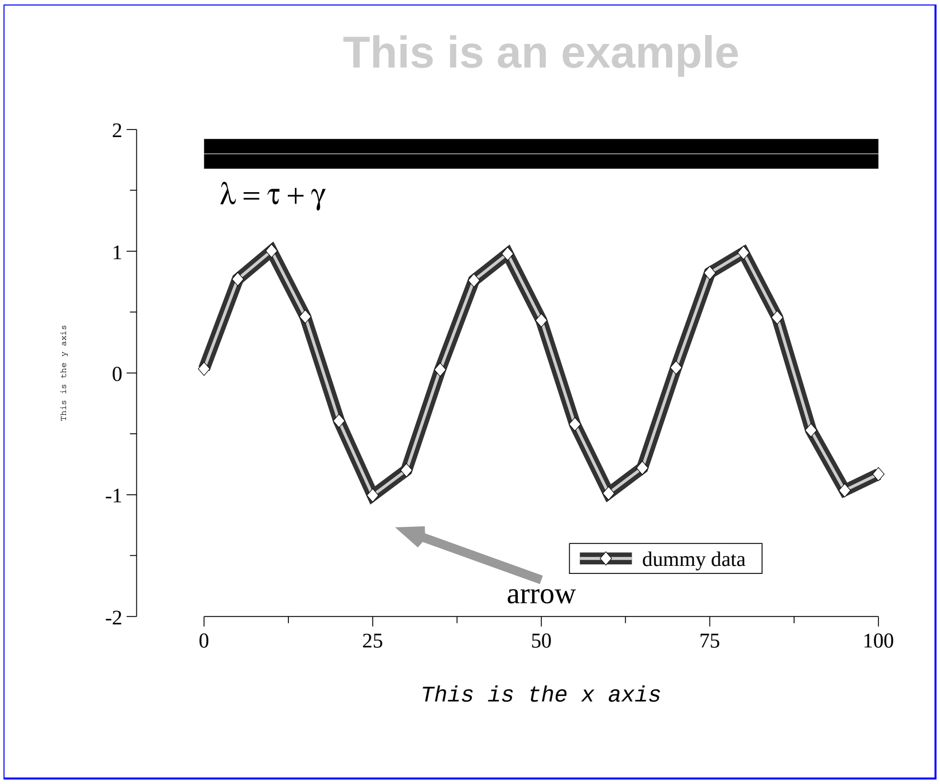


Figure 11.4: Font groups have been redefined to create many of the special effects in this example. See the main text for details.

Figure 11.4 shows several more complex uses of different font groups. This figure was produced using the command:

```
plt example14.data % -f example14.format
```

using the same data file as in the previous example. The format file, `example14.format`, contains:

```
# Set font and point size for font group "l" (labels).
sf l Fs,P20

# Define two custom font groups, "fontgroup1" and "fontgroup2".
sf fontgroup1 W40,G.2
sf fontgroup2 Fh-b,P30,G.8

# Plot the data three times, first with a broad dark line,
# then with a medium light line, finally with open diamonds
# (symbol 2).
p 0,1n(fontgroup1) 0,1n(W10,G.8) 0,1S2

# Print the plot title.
t (fontgroup2) This is an example

# Local specifications are given below for the x and y axis
# titles.  These would override defaults for the predefined
# title (-sf t ...) font group had any been set.
x (Fc-o,P15) This is the x axis
y (Fc,P6) This is the y axis

# The label uses font group l defaults set above unless local
# instructions are given.  The choice of font "s" (Symbol)
# above means that the Roman letters in the label string are
# rendered as Greek letters on the plot.
L .1 .87 - l = t + g

# Set y axis parameters.  By specifying the crossing point at
# x = -10, we place the y axis away from the data at x=0, and
# the axes do not actually touch each other.
ya -2 2 - - 2 -10
```

[`example14.format` continues on the next page]

[example14.format, continued]

```
# Construct the legend (key).  The final argument (.1) of the
# lp option lengthens the sample plot segment to 10% of the x
# axis length.  Three entries are overlaid on line 0 of the
# legend in order to construct a sample plot segment with the
# same appearance as in the data plot.
lp .65 .15 1.05 .1
le 0 0 dummy data
le 0 1
le 0 2

# Draw an arrow, using local specifications for the point size
# (determines the size of the arrowhead), grey level, and line
# width.
a (P30,G.6,W30) 30 -1.3 50 -1.7

# Label the arrow, overriding the default font so that the
# label is printed  using Roman rather than Greek characters.
l (Ft) 50 -1.8 - arrow

# Demonstrate line segment drawing using local specifications.
# First draw  a broad black line segment, then a narrow white
# line segment over the first.
c (W100,G0) 0 1.8 100 1.8
c (W2,G.99) 0 1.8 100 1.8
```

Figure 11.5 illustrates how `plt` can be used to produce plots in color. This plot was created using the command:

```
plt -f colors.format
```

The file `colors.format` contains:

```
X 0 1
Y 0 1
sf a CSkyBlue
sf red Cred
sf green Cgreen
sf blue Cblue
sf PapayaWhip CPapayaWhip
sf coral Ccoral
sf yellow Cyellow
sf custom1 C#007f00
sf custom2 C#cc007f
sf custom3 C#101010
sf white Fh-b Cwhite
sf black Fh Cblack
D (red) .1 .8 .3 1.0
L (white) .2 .9 CC red
D (green) .4 .8 .6 1.0
L (white) .5 .9 CC green
D (blue) .7 .8 .9 1.0
L (white) .8 .9 CC blue
D (PapayaWhip) .1 .5 .3 .7
L (black) .2 .6 CC PapayaWhip
D (coral) .4 .5 .6 .7
L (white) .5 .6 CC coral
D (yellow) .7 .5 .9 .7
L (black) .8 .6 CC yellow
D (custom1) .1 .2 .3 .4
L (white) .2 .3 CC #007f00
D (custom2) .4 .2 .6 .4
L (white) .5 .3 CC #cc007f
D (custom3) .7 .2 .9 .4
L (white) .8 .3 CC #101010
x (green) x axis
y (red) y axis
t (blue) Plotting in color
```

See appendix A, *Color Names*, beginning on page 91), for details on specifying arbitrary colors, and for a complete list of predefined colors.

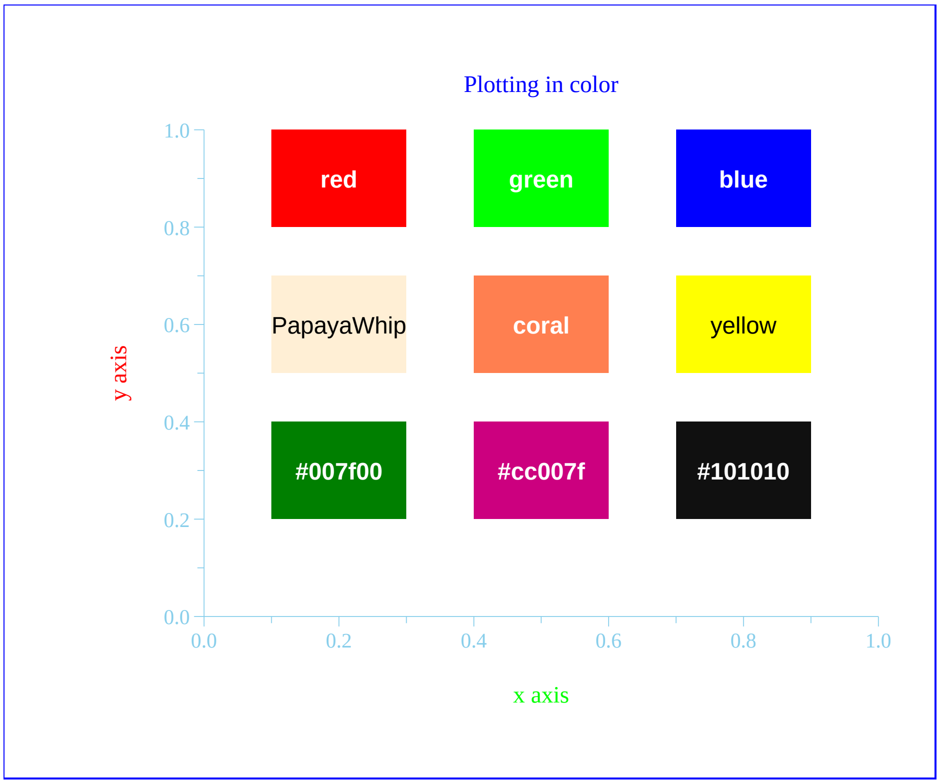


Figure 11.5: The color blocks are drawn using the `-D` (dark box) option. See the main text for details.

Chapter 12

Advanced Axis Options

This chapter introduces more options for manipulating and labelling your axes, to provide fine control over the appearance of your plots.

`-lx` ***base subticks***

`-ly` ***base subticks*** Use these options to create logarithmic x and y axes; *base* is the base of the logarithms (default: 10), and *subticks* is either `yes` or `no`. If the axis has a small number of major ticks, `plt` draws subticks by default; use the *subticks* argument to change `plt`'s default behavior.

The `-lx` and `-ly` options affect only how the axes are drawn; `plt` does not log-transform the input data, so you must provide the logarithms of the data to be plotted. Note that *base* does not have to be numeric; for example, "`-lx e`" will cause plt to draw a logarithmic x-axis with ticks labelled e^0, e^1, e^2, ... (or other integer powers of e as appropriate for the range of abscissas).

`-xe` ***xmin-error xmax-error***

`-ye` ***ymin-error ymax-error*** These options can be used to set the amounts by which the axis ranges are allowed to exceed the ranges of the data when `plt` determines the axis ranges automatically.

`-xm` ***tick-base***

`-ym` ***tick-base*** Make axis ticks a multiple of the specified *tick-base*.

`-xo` ***x-axis-offset***

`-yo` ***y-axis-offset*** These options allow the axes to be moved from their default positions; they are particularly useful if data would otherwise be obscured by axis markings. The `-xo` option moves the x axis down by *x-axis-offset*, which is expressed as a fraction of the y axis length; the `-yo` option moves the y axis left by *y-axis-offset*, which is a fraction of the x axis length (see figure 11.4).

`-xr`

`-yr` Use these options to plot the x axis at the top of the plot, or the y axis on the right side of the plot.

`-xt` ***x label tick-size***

`-yt` ***y label tick-size*** These options add an extra labelled tick at the specified x or y position on the corresponding axis. *label* can be any string; if omitted, the label is x or y, formatted in the same way as the other labelled axis ticks. *tick-size* specifies the length of the tick, as a multiple of the default length for labelled ticks; if omitted, *tick-size* is 1. Use a negative *tick-size* to change the direction of the tick and the placement of the label (the result depends also on the *grid-mode*; see `-g` below).

`-xts` ***x tick-size***

`-yts` ***y tick-size*** Use these options to force a labelled tick to be placed at the specified x or y positions on the axes, and to multiply the lengths of all ticks on the corresponding axis by the factor *tick-size* (other than any extra ticks generated using `-xt` or `-yt`). Use a negative *tick-size* to change the direction of the tick and the placement of the labels (the result depends also on the *grid-mode*; see `-g` below).

`-g` ***grid-mode ...*** The `-g` option accepts one or more *grid-mode* specifiers, which define how to create a grid. Note that if more than one *grid-mode* is supplied, they must be surrounded by quotes or separated by commas, because `-g` takes only one string as its argument. The defined *grid-modes* are:

`in`	puts ticks inside the grid
`out`	puts ticks outside the grid (default)
`both`	puts ticks inside and outside the grid
`none`	suppresses ticks
`sym`	make symmetric axes (at top and right)
`grid`	make a full grid (extend major ticks across the entire plot)
`xgrid`	extend major x axis ticks across the entire plot
`ygrid`	extend major y axis ticks across the entire plot
`sub`	make a fine grid (extend all ticks across the entire plot)

Figure 12.1 demonstrates several of these options, featuring logarithmic axes and a variety of grid modes. The data file, `ldemo.data`, is included in the `doc` directory of the `plt` distribution (see appendix F).

As a final example, figure 12.2 demonstrates use of the `O` plotstyle, the `-yo` option for shifting the y axis to the left, the `-xts option` to create an extra labelled tick mark, the `-size` option (see appendix B), relative parameter values in transient fontgroup specifications, overlays, and multiple plot windows. The figure was created using the following commands:

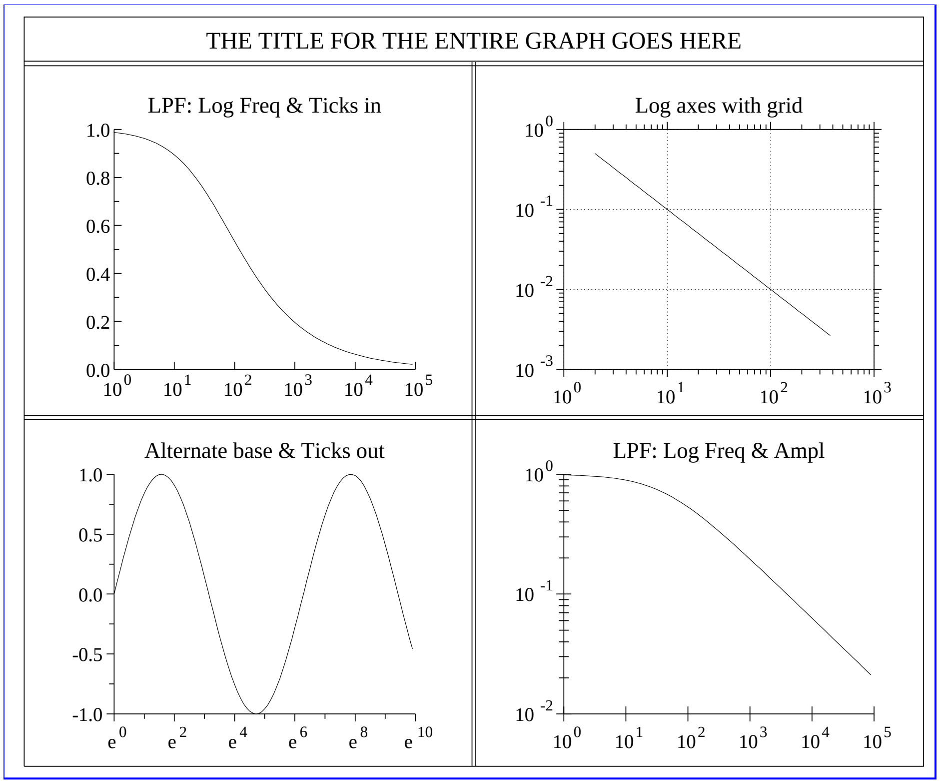


Figure 12.1: This example demonstrates several of the advanced axis options. It was produced using these commands:

```
plt -wq 0 -t"THE TITLE FOR THE ENTIRE GRAPH GOES HERE"
plt ldemo.data 2 3 -wqs 1 -lx -g in -t"LPF: Log Freq & Ticks in"
plt ldemo.data 4 5 -wqs 4 -lx -ly -g both -t"LPF: Log Freq & Ampl"
plt ldemo.data 0 1 -wqs 3 -lx e -g out  -t"Alternate base & Ticks out"
plt ldemo.data 6 7 -wqs 2 -lx -ly - yes \
 -g grid,sym,out -t"Log axes with grid"
```

```
plt power.data 0 1 2 3 4 5 -F"
size .7 5 5
sf p W1
W - .1 - .47
# P*1.5 means multiply point size by 1.5.  The point size for the 'O'
# plot type has no effect on the plot, but changes the height of the
# box in the legend.
p 0,1,5O(G.95,P*1.5) 0,2,4O(G.85)
lp .7 1.2 .9
le 0 0   5% Conf. Limits
le 1 1 25% Conf. Limits
t
y Power Error
yo 0.02
y Estimate/True LO Power
xts .85
xa .82 1 .01 - 5
x Fraction Available Data"

plt power.data 0 6 7 8 9 10 -F"
s ex
size .7 5 5
sf p W1
W - .53 - .9
p 0,1,5O(G.95,P*1.5) 0,2,4O(G.85)
t Confidence Limits for Spline Power Estimates
xts .85
xa .82 1 .01 - 5
y Estimate/True HI Power
yo 0.02"
```

The data file, `power.data`, is included in the `doc` directory of the `plt` distribution (see appendix F).

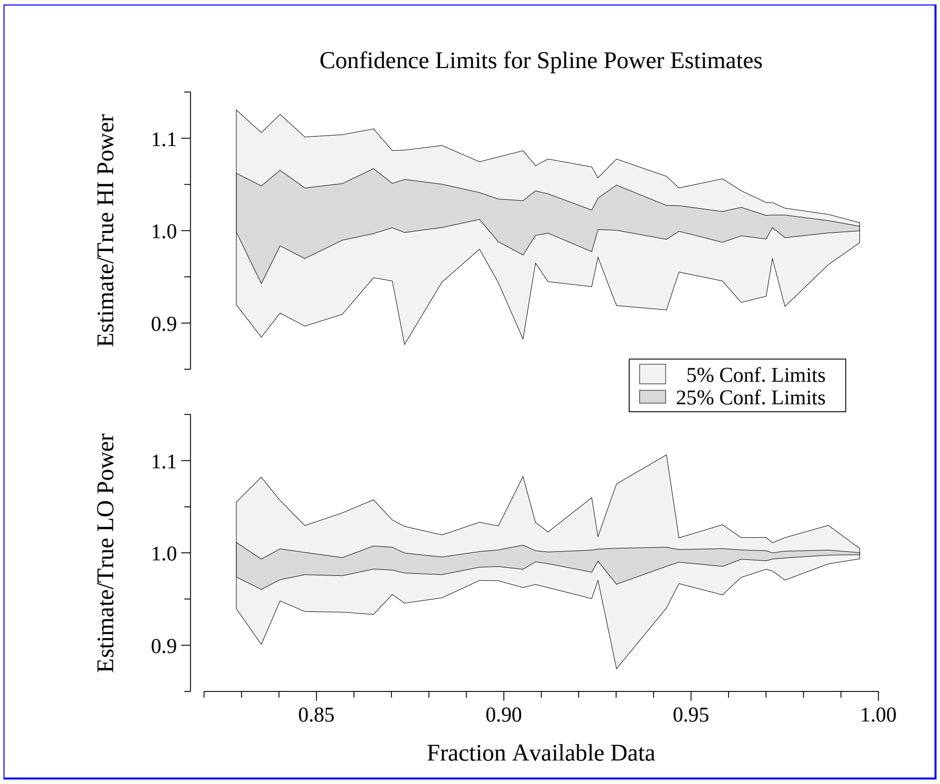


Figure 12.2: A complex plot (see text).

Appendix A

Color Names

Colors can be specified as hexadecimal triplets, in the form $\#rrggbb$, where rr, gg, and bb are the red, green, and blue components respectively. Thus, for example, `#0000ff` is fully saturated blue, `#000080` is dark blue, and `#ffff00` is bright yellow (100% red + 100% green).

Alternatively, named colors from the X11R6 color database can be used to specify a color. Note that case is not significant in color names (`blue`, `Blue`, `BLUE`, and `bLuE` are all acceptable names for the same color). These named colors are available for either X11 or PostScript plots:

```
AliceBlue
AntiqueWhite
AntiqueWhite1
AntiqueWhite2
AntiqueWhite3
AntiqueWhite4
aquamarine
aquamarine1
aquamarine2
aquamarine3
aquamarine4
azure
azure1
azure2
azure3
azure4
beige
bisque
bisque1
bisque2
bisque3
bisque4
green1
green2
green3
green4
GreenYellow
grey
honeydew
honeydew1
honeydew2
honeydew3
honeydew4
HotPink
HotPink1
HotPink2
HotPink3
HotPink4
IndianRed
IndianRed1
IndianRed2
IndianRed3
IndianRed4
ivory
orchid
orchid1
orchid2
orchid3
orchid4
PaleGoldenrod
PaleGreen
PaleGreen1
PaleGreen2
PaleGreen3
PaleGreen4
PaleTurquoise
PaleTurquoise1
PaleTurquoise2
PaleTurquoise3
PaleTurquoise4
PaleVioletRed
PaleVioletRed1
PaleVioletRed2
PaleVioletRed3
PaleVioletRed4
PapayaWhip
```

```
black               ivory1                PeachPuff
BlanchedAlmond      ivory2                PeachPuff1
blue                ivory3                PeachPuff2
blue1               ivory4                PeachPuff3
blue2               khaki                 PeachPuff4
blue3               khaki1                peru
blue4               khaki2                pink
BlueViolet          khaki3                pink1
brown               khaki4                pink2
brown1              lavender              pink3
brown2              LavenderBlush         pink4
brown3              LavenderBlush1        plum
brown4              LavenderBlush2        plum1
burlywood           LavenderBlush3        plum2
burlywood1          LavenderBlush4        plum3
burlywood2          LawnGreen             plum4
burlywood3          LemonChiffon          PowderBlue
burlywood4          LemonChiffon1         purple
CadetBlue           LemonChiffon2         purple1
CadetBlue1          LemonChiffon3         purple2
CadetBlue2          LemonChiffon4         purple3
CadetBlue3          LightBlue             purple4
CadetBlue4          LightBlue1            red
chartreuse          LightBlue2            red1
chartreuse1         LightBlue3            red2
chartreuse2         LightBlue4            red3
chartreuse3         LightCoral            red4
chartreuse4         LightCyan             RosyBrown
chocolate           LightCyan1            RosyBrown1
chocolate1          LightCyan2            RosyBrown2
chocolate2          LightCyan3            RosyBrown3
chocolate3          LightCyan4            RosyBrown4
chocolate4          LightGoldenrod        RoyalBlue
coral               LightGoldenrod1       RoyalBlue1
coral1              LightGoldenrod2       RoyalBlue2
coral2              LightGoldenrod3       RoyalBlue3
coral3              LightGoldenrod4       RoyalBlue4
coral4              LightGoldenrodYellow  SaddleBrown
CornflowerBlue      LightGray             salmon
cornsilk            LightGreen            salmon1
cornsilk1           LightGrey             salmon2
cornsilk2           LightPink             salmon3
cornsilk3           LightPink1            salmon4
cornsilk4           LightPink2            SandyBrown
cyan                LightPink3            SeaGreen
cyan1               LightPink4            SeaGreen1
```

cyan2
cyan3
cyan4
DarkBlue
DarkCyan
DarkGoldenrod
DarkGoldenrod1
DarkGoldenrod2
DarkGoldenrod3
DarkGoldenrod4
DarkGray
DarkGreen
DarkGrey
DarkKhaki
DarkMagenta
DarkOliveGreen
DarkOliveGreen1
DarkOliveGreen2
DarkOliveGreen3
DarkOliveGreen4
DarkOrange
DarkOrange1
DarkOrange2
DarkOrange3
DarkOrange4
DarkOrchid
DarkOrchid1
DarkOrchid2
DarkOrchid3
DarkOrchid4
DarkRed
DarkSalmon
DarkSeaGreen
DarkSeaGreen1
DarkSeaGreen2
DarkSeaGreen3
DarkSeaGreen4
DarkSlateBlue
DarkSlateGray
DarkSlateGray1
DarkSlateGray2
DarkSlateGray3
DarkSlateGray4
DarkSlateGrey
DarkTurquoise
DarkViolet
LightSalmon
LightSalmon1
LightSalmon2
LightSalmon3
LightSalmon4
LightSeaGreen
LightSkyBlue
LightSkyBlue1
LightSkyBlue2
LightSkyBlue3
LightSkyBlue4
LightSlateBlue
LightSlateGray
LightSlateGrey
LightSteelBlue
LightSteelBlue1
LightSteelBlue2
LightSteelBlue3
LightSteelBlue4
LightYellow
LightYellow1
LightYellow2
LightYellow3
LightYellow4
LimeGreen
linen
magenta
magenta1
magenta2
magenta3
magenta4
maroon
maroon1
maroon2
maroon3
maroon4
MediumAquamarine
MediumBlue
MediumOrchid
MediumOrchid1
MediumOrchid2
MediumOrchid3
MediumOrchid4
MediumPurple
MediumPurple1
MediumPurple2
SeaGreen2
SeaGreen3
SeaGreen4
seashell
seashell1
seashell2
seashell3
seashell4
sienna
sienna1
sienna2
sienna3
sienna4
SkyBlue
SkyBlue1
SkyBlue2
SkyBlue3
SkyBlue4
SlateBlue
SlateBlue1
SlateBlue2
SlateBlue3
SlateBlue4
SlateGray
SlateGray1
SlateGray2
SlateGray3
SlateGray4
SlateGrey
snow
snow1
snow2
snow3
snow4
SpringGreen
SpringGreen1
SpringGreen2
SpringGreen3
SpringGreen4
SteelBlue
SteelBlue1
SteelBlue2
SteelBlue3
SteelBlue4
tan
tan1

DeepPink
DeepPink1
DeepPink2
DeepPink3
DeepPink4
DeepSkyBlue
DeepSkyBlue1
DeepSkyBlue2
DeepSkyBlue3
DeepSkyBlue4
DimGray
DimGrey
DodgerBlue
DodgerBlue1
DodgerBlue2
DodgerBlue3
DodgerBlue4
firebrick
firebrick1
firebrick2
firebrick3
firebrick4
FloralWhite
ForestGreen
gainsboro
GhostWhite
gold
gold1
gold2
gold3
gold4
goldenrod
goldenrod1
goldenrod2
goldenrod3
goldenrod4
gray
green
MediumPurple3
MediumPurple4
MediumSeaGreen
MediumSlateBlue
MediumSpringGreen
MediumTurquoise
MediumVioletRed
MidnightBlue
MintCream
MistyRose
MistyRose1
MistyRose2
MistyRose3
MistyRose4
moccasin
NavajoWhite
NavajoWhite1
NavajoWhite2
NavajoWhite3
NavajoWhite4
navy
NavyBlue
OldLace
OliveDrab
OliveDrab1
OliveDrab2
OliveDrab3
OliveDrab4
orange
orange1
orange2
orange3
orange4
OrangeRed
OrangeRed1
OrangeRed2
OrangeRed3
OrangeRed4
tan2
tan3
tan4
thistle
thistle1
thistle2
thistle3
thistle4
tomato
tomato1
tomato2
tomato3
tomato4
turquoise
turquoise1
turquoise2
turquoise3
turquoise4
violet
VioletRed
VioletRed1
VioletRed2
VioletRed3
VioletRed4
wheat
wheat1
wheat2
wheat3
wheat4
white
WhiteSmoke
yellow
yellow1
yellow2
yellow3
yellow4
YellowGreen

In addition, 101 shades of grey can be specified using `grey0` or `gray0` (black) through `grey100` or `gray100` (white).

When producing Postscript plots, `plt` supplies the translation from color names names to RGB values, so the names above are always available. When producing X11 plots, however, the X server performs the translation; thus, if you are using an old or nonstandard X server, it is possible that a different set of color names may be recognized by the server. If you encounter problems, look for the file `rgb.h` in the directory that contains '`.h`' files associated with the X11 libraries; this file will contain the color name database for your X server.

Appendix B

Preparing Printed Output

As illustrated throughout this book, `plt` can generate high-quality printed output, as well as EPS (encapsulated PostScript) files that can be included in LaTeX and other documents.

`plt` does not include drivers for non-PostScript printers. If your printer does not accept PostScript, however, you may be able to convert `plt`'s PostScript output to your printer's printer language using `ghostscript` (a high-quality PostScript interpreter freely available from `http://www.ghostscript.com/`). `ghostscript` supports a wide variety of popular laser, ink-jet, and dot matrix printers, and can also convert PostScript into many popular image formats, including PNG, JPEG, TIFF (including fax formats), and PDF.

To understand how to make printed plots, a brief overview of how `plt` generates PostScript output may be helpful.

`plt` itself is a shell script that invokes a device-specific driver. Currently, these drivers are `xplt`, which draws into an X window created using `xpltwin`, and `lwplt`, which creates PostScript output. (The name `lwplt` comes from the Apple LaserWriter, which was the first commercially available PostScript printer.) Typically, you must use a command such as:

```
plt -T lw ... | lwcat -ps >plt-output.ps
```

to generate a complete PostScript file (in this example, `plt-output.ps`) suitable for printing. The use of the "`-T lw`" option selects the `lwplt` driver. (Some users prefer to select the driver by setting the `PTERM` environment variable rather than by a command-line option. Use whatever method you prefer.) The postprocessor, `lwcat`, is a shell script included with the `plt` package that appends its standard input (in this case, the output of `(lw)plt`) to a *prolog file* that contains definitions for the PostScript procedures used by `lwplt`. The name of the prolog file, usually `/usr/lib/ps/-plt.pro`, is specified by the `PROLOG` variable set near the beginning of `lwcat`.

If you wish to print the output immediately, you may omit the `-ps` option and the output redirection from the `lwcat` command:

```
plt -T lw ... | lwcat
```

In this simplified version, the plot is immediately sent to the print spooler for printing (under Unix, GNU/Linux, or Mac OS X).

`lwcat` offers several other options:

`-no`	spool the output, but don't eject the page (use this option if you wish to overlay the output with additional material produced by another program, such as `troff`)
`-psv`	collect Postscript in a temporary file and view with `gv` (`ghostscript`)
`-gv`	same as `-psv` (default if run from a Cygwin/bash window under MS-Windows)
`-eps`	write EPSF (encapsulated PostScript format) to the standard output (do not spool)
`-pdf`	write PDF to the standard output (see appendix C)
`-png`	write PNG to the standard output (see appendix C)

Note that the output produced using `-eps` is only a close approximation to EPSF; it is acceptable to LaTeX's `epsfig` package, at least.

By default, the output appears within a 6.75x6 inch window, the lower left corner of which is positioned 1 inch from the left edge and 3.5 inches from the bottom edge of the page. Additional `lwcat` options may be used to modify the size and location of the output:

`-landscape`	use landscape mode (rotate plot 90 degrees counter-clockwise)
`-sq`	plot in a 6x6 inch window, 1.25 inches from the left edge and 3.5 inches from the bottom edge
`-t`	plot in a 6.25x6.25 inch window, positioned as for `-sq`
`-t2`	plot in a 6.25x4 inch window, positioned as for `-sq`
`-CinC`	plot in a 4.75x3.15 inch window, positioned as for `-sq`
`-sq2`	plot in a 4.5x5.5 inch window, 2.5 inches from the left and bottom edges
`-v`	plot in a 7x9.5 inch window, 0.75 inches from the left and bottom edges
`-v2`	plot in a 7x8.5 inch window, positioned as for `-v`
`-va4`	plot in a 190x275 mm window, centered on an A4 sheet

Note the `-landscape` option changes the orientation of the *plot*, and not that of the *plot window*. Thus, for example, to print a landscape-format plot that covers the largest possible area on an 8.5x11 inch page, use both `-landscape` and `-v`; to cover the largest possible area on an A4 page with a landscape-format plot, use `-landscape` and `-va4`.

If none of these plot window sizes is what you want, there are several ways to define a custom plotting window for PostScript output:

1. Use `plt`'s `-W` or `-w` options, described earlier. (This is the only solution that works for screen plots as well as PostScript plots.)

2. Use `plt`'s `-size` option. This option is effective only when making a PostScript plot.

   `-size` *fscl width height left-margin bottom-margin*

   The *fscl* argument is a factor applied to the point sizes for all font groups, and is *independent* of the scaling applied to the plot. Thus, if *fscl* is 1, all text appears at the point sizes specified in the font group definitions, or the defaults. As another example, if *fscl* is 0.5, and the point size for the title font group is 20, the title text will be printed in 10 point type.

   The *width*, *height*, *left-margin*, and *bottom-margin* arguments are all given in units of *inches* (1 inch = 25.4 mm), and with reference to a portrait-format page (i.e., with the long edge vertical).

   Note that `plt`'s `-size` option is intended to be used *instead of* `lwcat`'s window definition options; if both are used, the `-size` option has no effect.

3. Define and export the environment variables `WDEF` or `LDEF` before invoking `lwcat`. `WDEF` is a string containing 5 numbers separated by spaces. The first number is a font scaling factor, defined as for `plt`'s `-size` option. The next two numbers are the x and y coordinates (in inches) of the lower left corner of the window, and the last two numbers are the x and y coordinates of the upper right corner. `LDEF` is defined similarly, but is used only in combination with `-landscape`; see the `lwcat` source for details.

4. Customize `lwcat`, following the models of the existing window definitions. Other window options can be easily added by adding additional `WDEF` and `LDEF` settings as described in comments within `lwcat`.

By default, `lwcat` prints a single copy of the `plt` output. Multiple copies can be produced using the options `-c2`, `-c3`, `-c4`, `-c5`, or `-c6`; this will almost always be much faster than rerunning `lwcat`, since the PostScript is downloaded and rasterized only once when using these options. To print more than 6 copies, repeat or combine these options as needed, e.g., to print ten copies, use `lwcat -c4 -c6 ...` (or `lwcat -c5 -c5`, etc.)

B.1 Generating EPSF using `plt`

To make `plt` produce an EPSF file suitable for inclusion in a LaTeX document, simply pipe the output of `plt` to `lwcat -eps`, adding any other `plt` or `lwplt` options you wish, and capture the output of `lwcat` in a file with the suffix `.ps` (this suffix is required by LaTeX's `epsfig` package). Note that `lwcat`'s `-c`*N* options are ignored when generating EPSF.

You may wish to print a "proof" copy of your `plt` figure. To do so, simply print the `.ps` file, for example, using the command

```
lpr file.ps
```

B.2 Including `plt` figures in a LaTeX document

To create a LaTeX document that includes `plt` figures, insert the line

```
\usepackage{epsfig}
```

at the top of the LaTeX source file, after the `\documentclass` statement.

Figure B.1 was generated using `plt -T lw ... | lwcat -eps >fig.ps`, and was included in this document using the commands

```
\begin{figure}
\begin{center}
\epsfig{file=fig}
\caption{File {\tt fig.ps}, ....}
\end{center}
\end{figure}
```

There is a problem here, because the figure doesn't fit between the LaTeX margins. Both the figure and the first line of this caption are inside a `center` environment, but the figure begins at the left margin. (There are empty areas at both the left and right edges of the figure.) The caption is properly centered between the margins, but is not centered with respect to the figure.

Figure B.1 was drawn in its "natural size", and appears at the same scale (though not at the same location on the page) as it would appear if `fig.ps` were printed by itself. Figure B.2 is also drawn from `fig.ps`, but it has been centered and scaled using the commands

```
\begin{figure}[h]
\begin{center}
\epsfig{file=fig,height=10cm}
\end{center}
\caption{This figure ....}
\end{figure}
```

The `epsfig` package allows a great deal of control over characteristics such as the aspect ratio (height and width may be scaled independently), the orientation, and the bounding box of the figure (which indirectly determines the amount of white space surrounding it). Figure B.3 illustrates how this can be done, using the commands

```
\begin{figure}
\begin{center}
\epsfig{file=fig,height=8cm,width=12cm,angle=90}
\end{center}
\caption{A rescaled, stretched, and rotated figure.}
\end{figure}
```

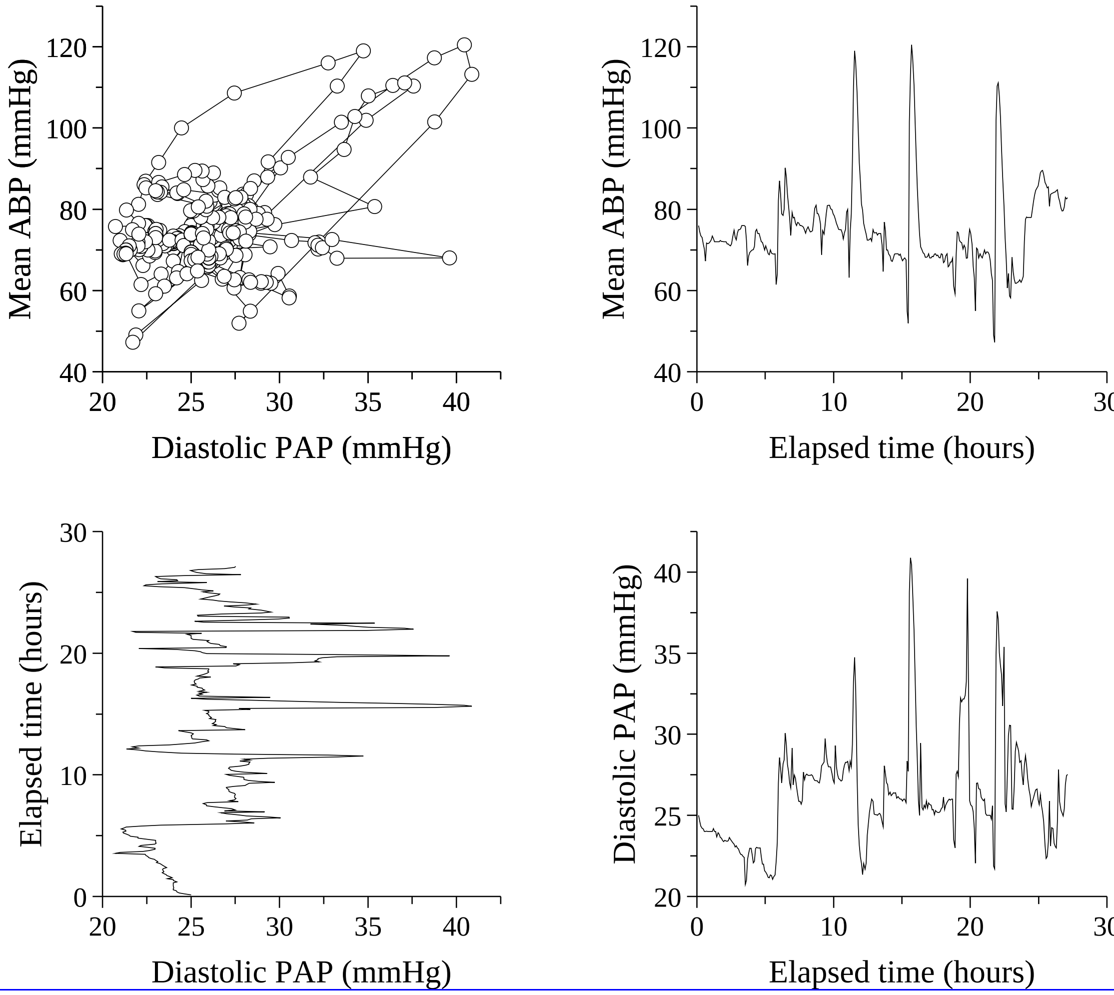


Figure B.1: File `fig.ps`, created using `plt` and included here using `epsfig`

.

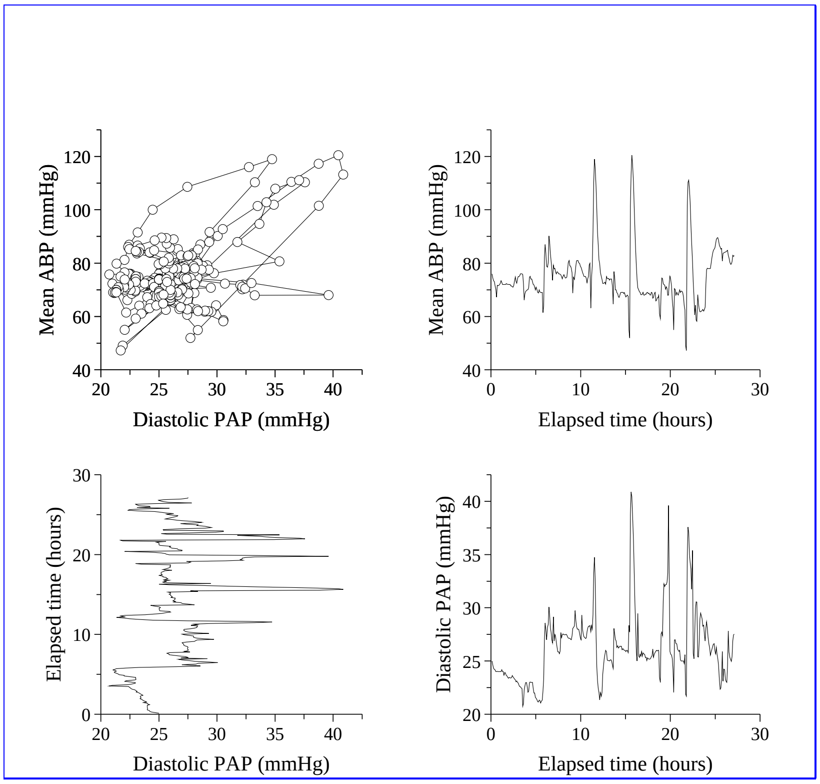


Figure B.2: This figure was also produced from `fig.ps`, but has been rescaled to a height of 10 cm. Note that this caption is right-justified, since it appears outside of the `center` environment, unlike the caption in figure B.1.

B.3 Margins, cropping, and bounding boxes

Every PostScript file included in a LaTeX document must have a defined bounding box. Normally, the bounding box is defined using a structured comment at the beginning of the PostScript file. For example, the PostScript source for the figures shown above begins like this:

```
%!PS-Adobe-3.0
%%BoundingBox: 90 252 540 684
```

In some cases, you may need to redefine the bounding box, either to adjust the amount of white space surrounding the figure or to crop your figure by selecting a clipping region. You may do so either by editing the PostScript file or in the LaTeX source file. Any bounding box definitions in the LaTeX source override those in the PostScript file. For details, see section 11.3, “Merging Text and PostScript Graphics”, in *The LaTeX Companion*, pp. 317–320.

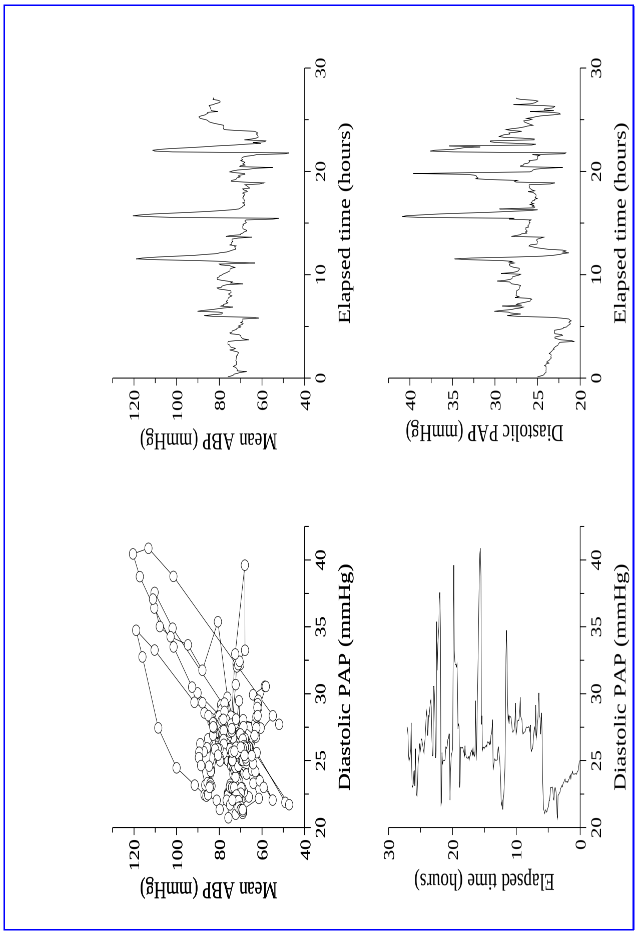
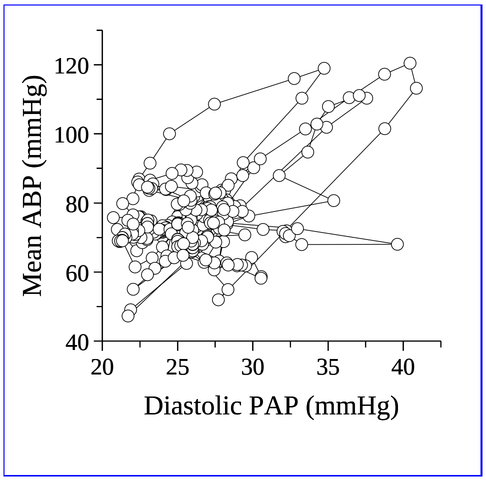


Figure B.3: A rescaled, stretched, and rotated figure.

An easy way to determine the correct bounding box is to view your figure using `ghostview`. Move the mouse pointer to the lower left corner of the figure and note the coordinates (displayed by `ghostview` immediately above the `File` button); these are the first two numbers in the BoundingBox line. Now move the mouse pointer to the upper right corner of the figure; the coordinates displayed are the last two numbers in the BoundingBox line.

Using this technique, we can find a bounding box for the upper left panel in `fig.ps`. To plot that panel only, as shown here



we can use the commands

```
\begin{center}
\epsfig{file=fig,height=6cm,bbllx=100,bblly=425,bburx=300,
        bbury=625,clip=}
\end{center}
```

(This plot does not “float” within the text, since it was not put into a `figure` environment; rather, it appears in the same place in the output as in the LaTeX source file.) Note that the “=” is required in the `clip=` option, although no value follows the “=”. Also note that the `clip=` option is ignored by pdfLaTeX (see this figure in the PDF version of this book).

In some cases, `plt` does not respect its own window definitions and may plot elements that fall outside of the bounding box. These elements will appear in the LaTeX output unless you choose to suppress them using the `clip=` option. Such elements are usually close to the bounding box, and no special treatment is required.

B.4 Processing, previewing and printing

LaTeX documents with included `plt` figures can be processed, previewed, and printed without special treatment. Typically, your LaTeX source file has a name of the form `myfile.tex`, and you use `latex` to produce a `.dvi` file:

```
latex myfile
```

Once you have a .dvi file, you can view it with xdvi, as in:

```
xdvi myfile
```

Recent versions of xdvi include a View PS button that toggles display of included PostScript figures. Note that some older versions of xdvi are not able to display rotated or anisotropically scaled plots.

Use dvips to create a printable PostScript file from the .dvi file:

```
dvips myfile
```

The PostScript file generated by dvips can be previewed using gv (under MS-Windows, GSView) or printed directly using lpr:

```
gv myfile.ps
lpr myfile.ps
```

B.5 Using plt with pdfLATEX

Most of the techniques described in this chapter for preparing PostScript output from LATEX documents and .eps format plt figures will work without changes if you use pdfLATEX to prepare PDF output from LATEX documents and .pdf format plt figures. If you are creating figures specifically for inclusion in a PDF document, use a command of the form

```
plt -T lw ... | lwcat -pdf >fig.pdf
```

to make a PDF figure. If you have already generated a PostScript figure, use a command such as

```
epstopdf fig.ps
```

to make fig.pdf from an existing fig.ps. (epstopdf is freely available from CTAN, http://www.ctan.org/.) It is usually not necessary to make any changes to an existing LATEX source document in order to format it using pdfLATEX; thus, for example, your document should still use the epsfig package, even though your included figures will be in PDF rather than EPS format. When you specify the names of the figure files, always use the form

```
\epsfig{file=fig}
```

If you avoid writing file=fig.ps or file=fig.pdf, then the correct version of your figure will be chosen automatically when formatting your document with latex and dvips or with pdflatex.

The only feature of epsfig described in this appendix that is not currently supported by pdfLATEX is the clip= option, which is ignored. If you are reading the PDF version of this book, the figure in section B.3 illustrates the results; you should avoid using the clip= option if you anticipate using pdfLATEX.

Using pdfLATEX to format myfile.tex is a one-step process:

$\boxed{\texttt{pdflatex myfile}}$

Unless there are errors, this command should produce `myfile.pdf`, which can be viewed using `gv`, `xpdf`, Acrobat, or any other PDF reader.

Appendix C

Preparing Plots for the Web

Using the `lwcat` postprocessor described in appendix B, you can easily prepare plots in PDF or PNG format for publication on the Web. PNG format is one of only three graphics formats that are viewable by all current graphics-capable web browsers (the others are GIF, which produces inferior results and is proprietary, and JPEG, which is designed for continuous-tone photographs and is unsuitable for line drawings).

Unlike PNG, PDF is a vector format, so that PDF plots can be magnified to view fine detail. Both open-source and proprietary (but freely available) PDF viewer plugins are available for many web browsers, but not everyone has installed a PDF plugin, and viewing PDF images is usually noticeably slower (because of the overhead involved in loading the plugin and rasterizing the image) than viewing PNG images.

Use `lwcat` as described in the previous appendix, with the option `-png` or `-pdf` to write PNG or PDF format to the standard output. Usually you will wish to capture the file by redirecting the standard output, as in these examples:

```
plt -T lw ... | lwcat -png >output.png
plt -T lw ... | lwcat -pdf >output.pdf
```

To prepare PDF output, `lwcat` uses `epstopdf` (freely available from CTAN, `http://www.ctan.org/`), a perl script that translates the PostScript generated by `plt` using GhostScript, a free PostScript renderer that runs on a vast variety of operating systems. If you have followed the instructions for installing `plt` on your system, you should have GhostScript already; if not, get it from `http://ghostscript-.com/`. If you need perl, you can get it from CPAN, `http://www.cpan.org/`.

To prepare PNG output, `lwcat` uses `pltpng`, a tiny shell script included with `plt`. `pltpng` in turn uses `convert`, a utility included in ImageMagick, a free set of image-processing tools and libraries that also runs on all popular operating systems. Most GNU/Linux distributions include ImageMagick, which is also available for other versions of Unix as well as Mac OS X and MS-Windows; if you don't have it already, get it from `http://www.imagemagick.org/`.

It's easy to add support for generating any of the other image formats (over 90 of them, as of April 2005) supported by `convert`. Use `pltpng` as a model, and add an option for the desired format to `lwcat`.

Appendix D

On-Screen Plots

This appendix describes how to get additional control over on-screen plots made under the X Window System. The X Window System is a standard component of GNU/Linux and virtually all current versions of Unix, including Mac OS X. A high-quality implementation of X is freely available for MS-Windows (Cygwin/X, from `http://www.cygwin.com/`). If you are not using the X Window System, the features described here are not available; see appendix B for information about making PostScript plots that can be viewed or printed using GhostScript and `gv` or GSView.

Classic versions of `plt` also included screen drivers for a variety of (mostly obsolete) graphics terminals. These non-X screen drivers are available in the `classic` directory of the `plt` source distribution, but they lack support for some of the advanced features of `plt`; they are not described further in this book.

All of the features described in this book are available when using `plt` under the X Window System to make on-screen plots. (Classic versions of `plt` did not support text scaling and rotation, grey scale and color plotting, polygon fills, line width and style (dotted, dashed, etc.), opaque-center outline symbols, or progressive display in quickplot mode, all of which are now supported under X.)

In normal use, there is little you will need to know specifically about making screen plots; simply use `plt` as described throughout this book, and you will get screen plots unless you have selected PostScript output using the methods described in the previous appendix. Your plots will appear in a window named `Plt Window`, which you can dismiss by typing an escape character (or `q`, or `x`) into the window. If you don't dismiss the `plt` window, your next plot will appear in the same window; if you use the `-o` or `-s e` option, the previous contents of the window will not be erased, and you will be able to overlay plots.

The `plt` window can be resized (using whatever controls your window manager provides, typically by clicking and dragging the window frame using the mouse), but its contents will not be rescaled; run `plt` again to obtain a plot sized to fit the window.

If you would like to create a new `plt` window while keeping an existing one intact, use the command:

```
xpltwin
```

You don't need to run `xpltwin` to create the first `plt` window; `plt` looks to see if one exists and invokes `xpltwin` to create one if necessary. If two or more `plt` windows exist, `plt` draws into the most recently created one.

`xpltwin` accepts an optional geometry specification, using this syntax:

`xpltwin -g` *width*x*height*+−*xoff*+−*yoff*

The `-g` option accepts a geometry string (in the same format used by many other X applications) to specify the initial size of the window, and (if the window manager cooperates, which many do not) the window placement on the screen. The geometry string may also be specified as the value of the `XPGEOMETRY` environment variable; the command-line option overrides the environment variable if both are provided. If the geometry is unspecified, the window will fill one quarter of the screen, and will be centered on the screen if possible. The geometry string can also be saved as an X resource named `xplt.geometry`; you can save your preferred size in your `.Xdefaults` resource file by adding an entry such as:

```
xplt.geometry: 800x600
```

The resource is overridden by either the `XPGEOMETRY` environment variable or a command-line geometry string if either is set.

It is also possible to set the names of your preferred fonts in your `.Xdefaults` file. The defaults are:

```
xplt.font_h:     "-*-helvetica-medium-r-*-*-%d-*-*-*-*-*-*-*"
xplt.font_hb:    "-*-helvetica-bold-r-*-*-%d-*-*-*-*-*-*-*"
xplt.font_ho:    "-*-helvetica-medium-o-*-*-%d-*-*-*-*-*-*-*"
xplt.font_hbo:   "-*-helvetica-bold-o-*-*-%d-*-*-*-*-*-*-*"
xplt.font_hob:   "-*-helvetica-bold-o-*-*-%d-*-*-*-*-*-*-*"
xplt.font_c:     "-*-courier-medium-r-*-*-%d-*-*-*-*-*-*-*"
xplt.font_cb:    "-*-courier-bold-r-*-*-%d-*-*-*-*-*-*-*"
xplt.font_co:    "-*-courier-medium-o-*-*-%d-*-*-*-*-*-*-*"
xplt.font_cbo:   "-*-courier-bold-o-*-*-%d-*-*-*-*-*-*-*"
xplt.font_cob:   "-*-courier-bold-o-*-*-%d-*-*-*-*-*-*-*"
xplt.font_t:     "-*-times-medium-r-*-*-%d-*-*-*-*-*-*-*"
xplt.font_tr:    "-*-times-medium-r-*-*-%d-*-*-*-*-*-*-*"
xplt.font_tb:    "-*-times-bold-r-*-*-%d-*-*-*-*-*-*-*"
xplt.font_ti:    "-*-times-medium-i-*-*-%d-*-*-*-*-*-*-*"
xplt.font_tbi:   "-*-times-bold-i-*-*-%d-*-*-*-*-*-*-*"
xplt.font_tib:   "-*-times-bold-i-*-*-%d-*-*-*-*-*-*-*"
xplt.font_s:     "-*-symbol-medium-r-*-*-%d-*-*-*-*-*-*-*"
```

The final part of the resource name (e.g., `tbi` in `xplt.font_tbi`) indicates the font name as supplied in the `F` parameter of the `-sf option`, such as `t-b-i` in this example. The values attached to these names should normally be scalable font names, but with `%d` at the position that corresponds to the font size. You can use non-scalable fonts, or you can force a scalable font to be used only at one size by providing font names without %d. Use the xfontsel or xlsfonts utilities provided with most versions of the X Window System to see what fonts are available on your system.

The plt window can be given a name other than Plt Window by setting the value of the environment variable XPLTWIN to the name you have chosen. For example, using the Bourne or a similar shell, you can create a window named "Power Spectrum" and plot into it using these commands:

```
XPLTWIN="Power Spectrum"
export XPLTWIN
plt spectrum.data 0 1 -f spectrum.format
```

If you use csh or a similar shell, use

```
setenv XPLTWIN "Power Spectrum"
```

instead of the "XPLTWIN=..." and "export XPLTWIN" commands above.

By giving multiple plt windows different names and then changing the value of XPLTWIN, you can redirect successive plots to any of these windows in any order.

Appendix E

Scripting with `plt`

Using shell scripts (batch files), it is possible to automate the process of making many similar plots using `plt`. This appendix briefly illustrates how `plt` can be used in scripts. General techniques for writing shell scripts are well beyond the scope of this book. (There are other books devoted to this subject; choose one that describes your shell, such as the O'Reilly book *Learning the bash shell*, by Newham and Rosenblatt. If you use MS-Windows, install the `bash` shell, included in the free Cygwin toolkit available from `http://www.cygwin.com/`. If you don't know what a shell is, begin by reading an introductory book on Unix or Linux.)

Often it is useful to preview a plot on-screen before (or instead of) producing a printed copy. Writing a script makes it easy to do this. For example, we might create a script named `foo`, to produce figure 7.5:

```
#! /bin/sh

plt $* -wm 0 -t "This is the main title for the plot"
plt $* example11.data 0 1 -wm 1 -t "Window 1"
```

The file `foo` containing this script should be put in a directory in your `PATH`, and should be marked as executable using the command:

```
chmod +x foo
```

When you run this script, any command-line arguments you give to `foo` are passed to `plt` (the shell replaces "$*" with these arguments). Thus the command "`foo`" runs `plt` with no arguments other than those listed above, but the command "`foo -T lw`" runs the same `plt` commands with the additional arguments "`-T lw`". The output in the second case will be a stream of PostScript code; to print the plot, pipe the output to `lwcat`, as in:

```
foo -T lw | lwcat
```

`plt` provides the `-dev` option to allow you to specify options to be processed only if a specific device type is being used for output. The `-dev` option can be useful in scripts, since it permits you to choose different options for screen plots and printed plots (for example, you might want to use color in screen plots and shades of grey in printed plots). The syntax is:

`-dev` ***device* "*options ...*"** The value of *device* can be either "`lw`" (for PostScript output) or "`xw`" (for X11 output). Following *device* should be a quoted string containing one or more options (with initial hyphens omitted, as in `-F` format strings). These options will have effect only if the same *device* is selected for output (by using `-T` *device* or by setting the value of the environment variable `PTERM` equal to *device*).

For example, consider this variant of the script above:

```
#! /bin/sh

plt $* -wm 0 -dev lw -t "This is the main title for the plot"
plt $* example11.data 0 1 -wm 1 -dev lw "sf p G.5" \
 -dev xw "sf p Cred"
```

If this script is run without any arguments (or with the arguments "`-T xw`"), the plot appears in an X11 window with the data in red; if it is run with the arguments "`-T lw`", and the output is piped to `lwcat`, the plot is printed with the data in 50% grey.

Appendix F

How to get and install `plt`

The latest version of `plt` is available as a gzip-compressed tar archive from PhysioNet. Download

```
http://www.physionet.org/physiotools/plt.tar.gz
```

using your web browser.

F.1 Compiling `plt` under Linux or Unix

`plt` should be easy to compile and to run on any version of Unix (or Linux). You will need to have a working X Window System installation in order to compile `xplt` and to use it to make screen plots, but `lwplt` can be used without X (and it should be possible to compile it under any POSIX-compliant environment).

If you plan to use `plt` without X, you may wish to use GhostScript as a screen viewer for `lwplt`. GhostScript is a free PostScript renderer that runs on a vast variety of operating systems; get it from `http://ghostscript.com/`.

To install `plt`:

- If you have GNU `tar`, unpack the source archive using:

  ```
  tar xfvz plt.tar.gz
  ```

  Otherwise, if you have another version of `tar`, unpack it using:

  ```
  gzip -d plt.tar.gz
  tar xfv plt.tar
  ```

- You should now have a directory named `plt`, containing installation instructions in `INSTALL` and a set of sources in subdirectory `src`. Read and follow the instructions in `INSTALL` if they differ from those outlined here.

- Enter the `plt` directory: make any needed changes to the site-dependent variables in `Makefile`; compile the `plt` sources; and install the executables:

```
cd plt/src
make install
```

You may need root permissions to do this successfully. If the `make` command fails, read the text file named `Makefile` in the `plt/src` directory to see how to proceed.

Add the directory in which `plt` has been installed (usually `/usr/local/bin`) to your PATH if necessary.

- To test the installation, try this:

```
make xdemo
```

If you have `gv` (included in most Linux distributions, and freely available from `http://wwwthep.physik.uni-mainz.de/p̃lass/gv/`), you can see the same demonstration in PostScript format by typing:

```
make psdemo
```

- (Optional) If you wish to generate PNG images using `lwcat -png` (see appendix C, install ImageMagick (free download from `http://www.image-magick.org/`) if you have not already done so. Most Linux distributions include ImageMagick already.

F.2 Compiling and Using `plt` under Mac OS X (Darwin)

`plt` has been successfully compiled under Mac OS X 10.3.9 (Darwin 7.9.0) and several earlier versions of Mac OS X.

You will need to obtain and install several packages of free software needed by `plt`. For those not available from Apple, we suggest using Fink (`http://fink.sourceforge.net/`), an easy-to-install collection of free Unix tools that have been ported to Mac OS X.

Before compiling `plt`, download and install:

- X11 for Mac OS X (included in current Mac OS X CD sets, or from `http://www.apple.com/downloads/macosx/apple/x11formacosx.html`)
- Apple's XCode developer tools, including the X11 SDK from XCode's Packages folder (included in current Mac OS X CD sets, or from `http://developer.apple.com/tools/xcode/`)
- `gv` (from Fink; sources from `http://wwwthep.physik.uni-mainz.de/ plass/gv/` and `http://www.ghostscript.com/`)
- ImageMagick (from Fink; sources from `http://www.imagemagick.org/`)

Now follow the instructions above for Unix or Linux.

F.3 Compiling and Using `plt` under MS-Windows

It is possible without much trouble to compile and use `plt` under MS-Windows. Do not attempt to use a commercial C compiler for this purpose, however. The only C compiler that is known to work is Cygwin `gcc` (which is freely available from `http://www.cygwin.com/` and comes accompanied by a complete set of development tools and other utilities including GNU `tar`, `make`, `bash`, `cat`, `dd`, `rm`, and `tr`).

- Download and install Cygwin. In the *Select Packages* dialog, you will see a collapsed tree view of the numerous packages available. Click the View button once so that the indicator to its right changes from “Category” to “Full”. You may resize the dialog box as needed until the package names are visible. To select a package for installation, click on its rotating selector (in the New column) to cycle through the available choices until a version number appears. In most cases, you should choose the highest version number from among those that are available (this is usually the first alternative offered). Be sure to choose at least the following packages for installation:

  bc Needed by `pltf` for function plotting.

  gcc Needed to compile `plt`.

  gv Needed to preview PostScript plots

  ImageMagick Needed to make PNG plots (for use on web pages, etc.)

  make Needed to install `plt`.

  xorg-x11-devel Needed to make screen plots

- Download `plt.tar.gz` using your web browser.
- Open a Cygwin terminal emulator window (the Cygwin installation will have created a desktop icon for this) and perform the remaining steps by typing into it.
- Follow the instructions in the Linux or Unix section above for unpacking the compressed archive of `plt` sources (`plt.tar.gz`).
- Compile and install `plt` by typing:

```
make install
```

- To test the installation, start the X server (e.g., using `startx`) if it is not running already, and then (from an X terminal window) type:

```
make xdemo
```

  To test the PostScript driver, try this:

```
make psdemo
```

Appendix G

What's New?

This appendix summarizes some of the changes that have been made in `plt` since 1999. It may be helpful to users who are familiar with earlier versions.

First, what has *not* changed: `plt` is almost completely compatible with previous versions. Scripts written for classic `plt` should continue to work without changes. The only exception is that, in classic `plt`, the default values of the window coordinates xw and yw, used by the `-hl`, `-vl`, and `-L` options, were .5 (corresponding to the center of the rectangle defined by the axes). These default values are now those of the `RC` point of the previous label.

This change was made so that labels can be concatenated (see chapter 8). This feature allows you to change fonts or use superscripts or subscripts within labels. The new text box coordinates `LN`, `CN`, and `RN` are particularly useful for creating subscripts.

`plt` now runs on all popular platforms, with both X11 and PostScript output available under GNU/Linux, Mac OS X, MS-Windows, and Unix.

Perhaps the most obvious change is full support for all `plt` features in X11 (screen) plots. This includes control of line width and style (dotted, dashed, etc.); grey levels and colors, including polygon fills; and font selection, sizing, and rotation. Rendering rotated text is not supported directly by X11, and the method implemented in `xplt` is unavoidably slow; but it works! The X11 driver also now supports progressive display in *quickplot* mode (see page 31).

`plt` now has full color support in X11 and PostScript, and uses the X11R6 color name database (even under MS-Windows) to make it easy to create color plots that can be rendered on-screen or on paper without changes.

The low-level input functions used by `plt` have been reimplemented, with the result that empty lines and comments are now acceptable in text data files, and that most CSV (comma-separated value) format data files are now readable by `plt`. This work also fixed an old bug that caused the first row of a data file to be lost if `plt` received its input from a pipe.

Also new are `lwcat`'s `-png` and `-pdf` options for producing output that can be published on the Web in PDF and PNG formats (see appendix C).

G.1 Known bugs

Several minor bugs that were present in earlier versions of `plt` have been fixed. Those that have not yet been fixed are listed here; if you know of any others, or if you find any incompatibilities between the current version of `plt` and a classic version, please let me (`george@mit.edu`) know!

- Text scaling in `xplt` does not match that in `lwplt`, with the result that there can be minor differences in the size and spacing of plot labels when rendered on-screen and on paper.
- This bug list is probably incomplete!

Appendix H

`man` Pages for the `plt` Package

This appendix is based on the Unix-style `man` pages included in the `plt` package. If the `man` utility is available on your system (it is a standard component of GNU/Linux, Mac OS X, and Unix, and is also part of the Cygwin package for MS-Windows), you should be able to view this material at any time using the command “`man plt`” (or “`man lwcat`”, “`man imageplt`”, or “`man pltf`”).

`plt`

Name

plt - make 2-D plots

Synopsis

`plt` [*data-spec*] [*data-file*] [[*xcol*] *ycol*] [*options* ...] [`-T lw` | `lwcat` [*lwcat-options*]]

Description

This man page is intended as a supplement to the command-line help provided by `plt` itself (using the `-h` option, see below). If you have not previously used `plt`, please look at the *plt Tutorial and Cookbook*, which is included in the `plt` package (see `SOURCES` below).

`plt` is a non-interactive (command line-driven) plotting utility. `plt` can produce publication-quality 2D plots in PostScript from easily-produced text or binary data files, and can also create screen plots under the X Window System.

All data presented to `plt` must be organized in rows and columns. Columns are numbered beginning with zero, and each column contains values for a variable that can be used as an abscissa (x coordinate), ordinate (y coordinate), or (with appropriate options described below) a grey level, color, or other plot attributes. Rows are numbered beginning with one, and each row contains a value for each column. Within a *data-file*, values are always arranged in row-major order (all elements of row 1, followed by all elements of row 2, etc.).

Usually, data must be in text form in order for `plt` to read them. Each non-empty, non-comment line (row) in the input should contain a value for each column that will be plotted; any additional values or other extra text at the end of a row will be ignored. Columns can be separated by any number of spaces or tabs. Commas and single or double quotation marks can also be used as column separators with current versions of `plt`, though not with older versions. It is not necessary to line up the values in each row. There may also be spaces or tabs at the beginning of a line, and these will also be ignored.

If no *data-file* is specified, `plt` reads data from its standard input. The command-line arguments *xcol* and *ycol* specify the column numbers for the abscissas and ordinates respectively. If only one column number is specified, it is taken as *ycol*, and `plt` generates a series of abscissas automatically. If the *data-file* contains no more than two columns, both *xcol* and *ycol* may be omitted.

By default, `plt` reads all rows of the *data-file* and scales the x and y axes so that all data can be plotted. An optional *data-spec*, a string beginning with a colon (:), can be used to select a subset of the rows in the *data-file*. For details on using a *data-spec*, and for information about reading binary data files using `plt`, see the *plt Tutorial and Cookbook*.

`plt` recognizes a large number of *options* for controlling and customizing plots. To see a summary of all options, run "`plt -h`"; if this command is followed by one or more strings (which should not begin with hyphens), `plt` prints one-line summaries of all options beginning with those strings only.

`plt` can read its options from command-line arguments, from a *format file* (specified using the `-f` option), or from a *format string* (supplied on the command line, following the `-F` option). When using format files or format strings, omit the hyphen (-) before each option.

Options

Following is a brief summary of `plt`'s options. Note that many options require arguments. `plt` chooses a suitable default for most such arguments if the argument is supplied as '`-`'. See the *plt Tutorial and Cookbook* for further details.

`-p` ***plot-styles*** Specify style(s) for data plots. Available *plot-styles* include '`c`', '`C`', '`e+`*c*', '`e-`*c*', '`e:`*c*', '`E+`*n*', '`E-`*n*', '`E:`*n*', '`f`', '`i`', '`l`', '`m`', '`n`', '`N`', '`o`', '`O`', '`s`*c*', '`S`*n*', and '`t`'.

`-s` ***elements*** Suppress *elements* of output. Elements that can be suppressed include '`e`' (erasing the screen or beginning a new page before plotting), '`a`' (anything associated with axes), '`x`' (anything associated with the x axis), '`y`' (anything associated with the y axis), '`g`' (the grid), '`m`' (x and y axis tick marks), '`n`' (x and y tick mark numbers), '`t`' (x and y axis labels and plot title), '`l`' (user-supplied labels), '`p`' (data plots), and '`f`' ("figures" – boxes, line segments, arrows, and legends). In addition, these *elements* modify the effects of any other elements that follow: ‘`X`’ (restrict effects to x axis), ‘`Y`’ (restrict effects to y axis), and ‘`A`’ (apply effects to both axes); and the element ‘`C`’ reenables all elements.

`-X` ***xmin xmax*** Set the x-axis range (see also `-xa`).

`-Y` ***ymin ymax*** Set the y-axis range (see also `-ya`).

`-t` ***title*** Set the title for the plot (enclose *title* in quotes if it contains whitespace or begins with ‘`(`’ or ‘`[`’).

`-T` ***type*** Specify the output *type*, which may be `xw` (X11 window, the default under Unix or Linux and not available under MS-Windows), or `lw` (PostScript, the default under MS-Windows).

`-g` ***grid-mode*** Specify the grid style, which may be `in`, `out` (default), `both`, `none`, `sym` (make symmetric axes at top and right), `grid` (extend major ticks across the entire plot), `xgrid`, `ygrid`, or `sub` (extend all ticks across the entire plot).

`-h` **[** ***option-prefix ...***] Show help on options beginning with *option-prefix* (which should not begin with ‘`-`’). If *option-prefix* is omitted, show help on all options.

Within the next group of options, those with upper-case names (‘`-A`’, ‘`-B`’, ...) use *window coordinates* between (0,0) and (1,1); those with lower-case names (‘`-a`’, ‘`-b`’, ...) use *data coordinates*.

`-a` ***x0 y0 x1 y1*** Draw an arrow to *(x0,y0)* from *(x1,y1)*.

`-A` ***xw0 yw0 xw1 yw1*** Draw an arrow to *(xw0,yw0)* from *(xw1,yw1)*.

`-b` ***x0 y0 x1 y1*** Draw a box with opposite corners at *(x0,y0) and (x1,y1)*.

`-B` ***xw0 yw0 xw1 yw1*** Draw a box with opposite corners at *(xw0,yw0) and (xw1,yw1)*.

`-c` ***x0 y0 x1 y1*** Connect points *(x0,y0)* and *(x1,y1)*.

`-C` ***xw0 yw0 xw1 yw1*** Connect points *(xw0,yw0)* and *(xw1,yw1)*.

`-d` ***x0 y0 x1 y1*** Draw a dark (filled) box with opposite corners at *(x0,y0) and (x1,y1)*.

`-D` ***xw0 yw0 xw1 yw1*** Draw a dark (filled) box with opposite corners at *(xw0,yw0) and (xw1,yw1)*.

`-l` ***x y tbc label-string*** Print *label-string* at *(x,y)*. The *tbc* argument is a two-character text box coordinate that specifies how the label is to be positioned relative to *(x,y)*; the default (`CC`) centers the string at *(x,y)*.

`-L` ***xw yw tbc label-string*** As for `-l`, but using window coordinates *(xw,yw)*.

`-W` ***configuration subwindow*** Confine the plot to a predefined window, specified by the arguments. *configuration* specifies the number of subwindows (panels), using one of ‘`m`’ (1), ‘`b`’ (2), or ‘`q`’ (4), and *subwindow*’ specifies which panel is to be plotted (0 or 1 for ‘`m`’; 0, 1, or 2 for ‘`b`’; or 0, 1, 2, 3, or 4 for ‘`q`’). In each case, subwindow 0 creates the frame of the entire plot, and the other subwindows refer to regions where data can be plotted. Use this option with '`-o`' or '`-s e`' to create multi-panel plots in stages without starting a new page or erasing the window before starting each new stage.

`-W` ***xp0 yp0 xp1 yp1*** Define the region of the page in which to plot. The arguments are *page coordinates*; the page coordinates (0,0) and (1,1) correspond to the lower left and upper right corners of the page.

`-f` ***format-file*** Read options from the specified *format-file*.

`-fa` ***format-file*** Record the current axis parameters as options in the specified *format-file* (for use with a later `plt` command). The previous contents of *format-file*, if any, will be overwritten.

`-F` ***format-string*** Read options from the specified *format-string*.

`-o` Suppress all output except data plots.

`-cz` ***xfrom xincr*** Generate abscissas, beginning with *xfrom* (default: 0) and incrementing by *xincr* (default: 1) at each step.

`-ex` Don't exclude points outside axis limits.

`-hl` ***x y tbc n file*** Print the next *n* (default: 1000) lines of the specified *file* as a label, placing the reference point for the first line of the label at data coordinates *(x,y)*. The *tbc* argument is defined as for `-l` and is applied to each line of the label. The *file* is opened when first used by `-hl` or `-vl`, and remains open, so that successive `-hl` or `-vl` options referring to the same *file* read and print successive lines. At most `MAXLABELFILES` (defined in `plt.h`, currently 6) *file*s of label strings can be open at once.

`-vl` ***x y tbc n file*** As for `-hl`, but print the label in a vertical orientation (rotated 90 degrees counterclockwise).

`-le` ***linenumber plotnumber* [*text*]** Define the specified *linenumber* in the legend (see also `-lp`). Line numbers in the legend begin with 0 (the top line); plot numbers also begin with 0 (these refer to the data plots, and are used here to determine the line style for the entry's sample plot segment). The *text* is printed to the right of the sample plot segment. To create an entry with more than one line of text, use additional `-le` options with different *linenumber*s as necessary, omitting the *plotnumber* (use '`-`') for all but the first. If the same data are plotted more than once in a single figure to create an overlay (for example, using symbols over line segments), an overlaid legend entry can be created using additional `-le` options with the same *linenumber* and different *plotnumbers*, omitting the *text* for all but the first.

`-lp` ***xw0 yw0* [*boxscale* [*seglength* [*opaque*]]]** Define the window coordinates *(xw0, yw0)* of the upper left corner of the plot legend text, and other attributes for the plot legend (key). `plt` determines the size of the box it draws around the legend, but the calculated width of the box is multiplied by *boxscale*. The *seglength* option specifies the length of the sample plot segments, as a fraction of the x-axis length (default: 0.05). If *opaque* is '`yes`' (default), the background of the legend is opaque white; otherwise, the background is transparent (any previously drawn material remains visible through the legend box). Unless a `-lp` option is provided, no legend is printed.

`-lx` [*base* [*subticks*]] Draw a logarithmic x-axis; *base* is the base of the logarithms (default: 10), and *subticks* is either '`yes`' or '`no`'. If the axis has a small number of major ticks, `plt` draws subticks by default; use the *subticks* argument to change `plt`'s default behavior.

`-ly` [*base* [*subticks*]] Draw a logarithmic y-axis.

`-tf` *file* [*tbc*] Load the text string array from the specified *file*. Each line of the *file* defines an element of the string array; using plot styles `c` or `t`, these strings can be plotted in the same manner as data points. The optional *tbc* specifies how the positions of the strings are to be modified when they are printed, in the same way as for `-l`; by default, the strings are centered on the coordinates specified for them.

`-ts` "*string0 string1 ...*" [*tbc*] Load the text string array from the quoted argument (whitespace separates strings in the array) rather than from a file; otherwise, this option is the same as `-tf`.

`-fs` "*string0 string1 ...*" Load the font string array from the quoted argument. Using appropriate plot style (`-p`) options, the strings can be used to change the font, line style (solid, dotted, dashed, etc.), or drawing color.

`-x` *string* Set the x-axis title to *string* (which must be quoted if this option is used on the command line or if *string* begins with '(' or '[').

`-xa` *xmin xmax tick fmt tskip ycross* Specify the x-axis range (as *xmin* to *xmax*); the interval between x-axis tick marks; the format, *fmt*, in which to print the numbers (e.g., "`%.3f`", "`%.2e`"; any format that printf(3) can use for printing floating-point numbers is acceptable); the number of ticks per labelled tick, *tskip*; and *ycross*, the point on the y-axis that the x-axis should cross, in y-units. Any of these parameters may be supplied as "-", which causes `plt` to choose a reasonable value based on the input data.

`-xe` *xmin-error xmax-error* Use this option to specify the amount by which the x-axis range is allowed to exceed the range of x-values in the input data, when `plt` determines the x-axis range automatically.

`-xm` *tick-base* Make x-axis ticks be multiples of the specified *tick-base*.

`-xo` *x-axis-offset* Move the x-axis down by *x-axis-offset* (expressed as a fraction of the y-axis length).

`-xr` Draw the x-axis at the top of the plot

`-xt` ***x label* [*tick-size***] Add an extra labelled tick at the specified *x* position, and label it with the specified *label* (which may be any string). The optional *tick-size* argument specifies the length of the added tick, as a fraction of the default length for labelled ticks (e.g., a value of 1.5 makes the added tick 50 longer than the standard size).

`-xts` ***x* [*tick-size***] Force a labelled tick to appear on the x-axis at the specified *x* (the positions of the other labelled x-ticks are adjusted accordingly). *tick-size* is defined as for `-xt`.

`-y` ***string*** Set the y-axis title to *string* (see `-x`).

`-ya` ***ymin ymax tick fmt tskip xcross*** Set up the y-axis (see `-xa`).

`-ye` ***ymin-error ymax-error*** Set the allowable error in the y-axis range (see `-xe`).

`-ym` ***tick-base*** Make y-axis ticks be multiples of the specified *tick-base*.

`-yo` ***y-axis-offset*** Move the y-axis to the left by *y-axis-offset* (expressed as a fraction of the x-axis length).

`-yr` Draw the y-axis at the right edge of the plot.

`-yt` ***y label* [*tick-size***] Add an extra labelled tick at the specified *y* position (see `-xt`).

`-yts` ***y* [*tick-size***] Force a labelled tick to appear on the y-axis at the specified *y* (see `-xts`).

`-dev` ***pterm option*** Process *option* only if the value of `PTERM` is *pterm*. The `-dev` option may be useful in scripts that produce screen or printed plots in different formats.

`-sf` ***name specification*** Create a new font group with the specified *name* and set its specifications (font, point size, color/grey level, line width, and line style). See the chapter titled *Colors, Line Styles, and Fonts* in the *plt Tutorial and Cookbook* for details.

`-ch` ***height-factor width-factor*** Modify the height and width of all characters printed in the plot by the specified factors.

`-size` ***fscl width height left-margin bottom-margin*** Specify the size and position of the plot on the page. The *width*, *height*, *left-margin*, and *bottom-margin* are specified in *inches* (1 inch = 25.4 mm). *fscl* is a factor applied to the point size of all printed characters, *independently* of the scaling applied to the rest of the plot. This option is effective for printed plots only.

Screen and printed plots

By default, `plt` makes an X11 screen plot. To make a printed plot, use the option `-T lw`, and pipe the output of `plt` to lwcat`(1)`. Under Unix, GNU/Linux, or MacOS/X, `lwcat` uses the standard `lpr` print spooler to send `plt`'s output in PostScript format to the default printer. When running with a Cygwin/bash window under MS-Windows, or when using `lwcat`'s `-gv` option under Unix or Linux, the PostScript output is displayed on-screen using GhostScript (`GSView` under MS-Windows, or `gv` otherwise; these programs can save the output in a file or send it to a printer).

Examples

Create a text file with the following contents:

```
0 0 0
1 1 1
2 4 8
3 9 27
4 16 64
```

and call the file *powers*. Plot the first column vs. the second by:

```
plt powers 0 1 -t "Squares of small integers" -x "Integer"
-y "Square"
```

The same file can be used to generate a number of different plots, by choosing different columns. To plot the third column vs. the first, try:

```
plt powers 2 0 -t "Marshmallows" -x "Mass (kg)" -y "Height
(m)"
```

See Also

imageplt`(1)`, lwcat`(1)`, pltf`(1)`

The *plt Tutorial and Cookbook* (a book-length introduction to `plt`, included in the `plt` source package) contains many more examples.

Availability

`plt` is available as part of PhysioToolkit (see `SOURCES` below) under the GPL.

Authors

`plt` was originally written by Paul Albrecht, and is currently maintained by George B. Moody (`george@mit.edu`).

Sources

`http://www.physionet.org/physiotools/plt/`

`lwcat`

Name

lwcat - postprocess output of plt to make PostScript, EPS, PDF or PNG

Synopsis

`plt -T lw` ... | `lwcat` [*options* ...]

Description

`lwcat` collects the PostScript output of `plt`(1) and adds a prolog and epilog to create a complete PostScript document (or another format, if appropriate options have been selected). It is possible to concatenate the outputs of two or more `plt` runs to be processed as a single job by `lwcat`; see the *plt Tutorial and Cookbook* for details.

Output format

By default, `lwcat` sends its output directly to the default printer via `lpr`. These options may be used to modify this behavior:

`-no` Send the output to the printer, but don't eject the page (use this option if you wish to overlay the output with additional material to be produced by another program).

`-ps` Write PostScript to the standard output (not to the printer).

`-psv` Write PostScript to a temporary file and view it with `gv` (ghostscript).

`-gv` Same as `-psv`.

`-eps` Write EPSF (encapsulated PostScript format) to the standard output. Note that this is only a close approximation to EPSF; it is acceptable to LaTeX's epsfig package, at least.

`-pdf` Write PDF (portable document format) to the standard output.

`-png` Write PNG (portable network graphics) format to the standard output.

Window options

By default, the output appears within a 6.75x6 inch (171x152 mm) window, the lower left corner of which is positioned 1 inch (25.4 mm) from the left edge and 3.5 inches (89 mm) from the bottom edge of the page. The following options may be used to modify the size, location, and orientation of the output:

`-landscape` Use landscape mode (rotate plot 90 degrees counterclockwise).

-sq Plot in a 6x6 inch (152x152 mm) square window, 1.25 inches (32 mm) from the left edge and 3.5 inches (89 mm) from the bottom edge of the page.

-t Plot in a 6.25x6.25 inch (159x159 mm) square window, positioned as for `−sq`.

-t2 Plot in a 6.25x4 inch (159x102 mm) window, positioned as for `−sq`.

-CinC Plot in a 4.75x3.15 inch (121x80 mm) window, positioned as for `−sq`.

-sq2 Plot in a 4.5x5.5 inch (114x140 mm) window, 2.5 inches (63 mm) from the left and bottom edges of the page.

-v Plot in a 7x9.5 inch (178x241 mm) window, 0.75 inches (19 mm) from the left and bottom edges of the page (centered on a US letter sheet).

-v2 Plot in a 7x8.5 inch (178x216 mm) window, positioned as for `−v`.

-va4 Plot in a 190x275 mm window, centered on an A4 sheet.

-full Plot in a 7.5x10 inch (191x254 mm) window, centered on a US letter sheet.

-slide Plot in a 7.5x5 inch (191x127 mm) window, 0.5 inches (12.7 mm) from the left edge and 3 inches (76 mm) from the bottom edge of the page (3:2 aspect ratio, as for 35 mm slides).

-screen Plot in a 7.5x5.625 inch (191x143 mm) window, 0.5 inches (12.7 mm) from the left edge and 2.375 inches (60 mm) from the bottom edge of the page (4:3 screen aspect ratio).

-golden Plot in a 7.5x4.635 inch (191x118 mm) window, 0.5 inches (12.7 mm) from the left edge and 3.365 inches (85 mm) from the bottom edge of the page (the aspect ratio is approximately the ”golden ratio”, (1+sqrt(5))/2 = 1.61803

Other window options can be easily added; see the source for `lwcat` for details.

Copies

By default, `lwcat` prints a single copy. Multiple copies can be produced using the options `−c2`, `−c3`, `−c4`, `−c5`, and `−c6`; when using a PostScript printer, this will almost always be much faster than rerunning `lwcat`, since the document is downloaded and rasterized only once when using these options. To print more than 6 copies, repeat or combine these options as needed.

Files

/usr/lib/ps/plt.pro PostScript prolog

See Also

`plt`(1)

Availability

`lwcat` is available as part of the `plt` package in PhysioToolkit (see `SOURCE` below) under the GPL.

Authors

Paul Albrecht and George B. Moody (`george@mit.edu`)

Source

`http://www.physionet.org/physiotools/plt/src/lwcat`

imageplt

Name

imageplt - plot a greyscale image

Synopsis

`imageplt -d` *nrows ncols* [*options* ...] [*file*]

Description

`imageplt` provides a simple way to plot a greyscale image using `plt`(1). The required arguments, *nrows* and *ncols*, specify the numbers of rows and columns in the image. The input *file* (or the standard input, if no input file is specified) contains only the grey levels for each pixel (0 = white, 1 = black). Each entry is an ASCII-coded decimal floating point number, separated from adjacent entries by whitespace (one or more spaces, tabs, or newlines). The first *nrows* entries are the grey levels for column 0 of the image, botttom to top, and each successive column from left to right of the image follows. If *nrows* is small, it may be convenient to arrange the image file in columns and rows corresponding to those of the image, but this is not necessary. In no case should the length of a line of input exceed 50000 bytes (defined as MAXLEN in the source).

Options include:

-n Generate a negative image (1 = white, 0 = black).

-x *xmin xmax* Specify the range of the x-coordinates (default: *xmin*=0, *xmax*=*nrows*-1).

-y *ymin ymax* Specify the range of the y-coordinates (default: *ymin*=0, *ymax*=*ncols*-1).

The output of *imageplt* is text in three columns, to be plotted using the `-pc` option of `plt`, as in:

imageplt -d 10 10 foo | plt 0 1 2 -pc

See Also

`plt`(1), `pltf`(1)

Availability

`imageplt` is available as part of the `plt` package in PhysioToolkit (see `SOURCES` below) under the GPL.

Author

George B. Moody (george@mit.edu)

Source

http://www.physionet.org/physiotools/plt/plt/misc/imageplt.c

pltf

Name

pltf - make function plots

Synopsis

pltf [*expression* [*xmin* [*xmax* [*xinc*]]]]

Description

pltf provides a simple way to use bc(1) and plt(1) to generate plots of many common functions of a single variable. The command-line arguments are interpreted according to their position; pltf asks for values for any missing arguments.

The first argument, *expression*, can be any expression valid as input to bc(1), with the additional feature that the variable x may appear anywhere in the expression where a number would be allowed by bc. Some examples of valid expressions are:

x^3+3*x^2+3*x+1

(x + 1)^3

s(sqrt(x^2))

The first two of these are equivalent; note that whitespace and parentheses are allowed in expressions, although it is necessary to enclose such expressions in double quotes (e.g., "(x + 1)*e(x)") when entering them as command-line arguments in order to protect them from the shell. The last expression is the sine of the square root of x squared; see bc(1) for a complete list of available special functions, or invoke pltf with no command-line arguments to obtain a list.

The second and third arguments specify the domain of the function (the values over which x should vary), and the fourth argument specifies the x-increment (the difference between consecutive values of x for which the expression is to be evaluated).

pltf is a shell script that uses a helper application, ftable, to prepare input for bc -l. Invoke ftable directly (using the same arguments as for pltf) if you need to change the format of the plot or make a printed version of it. See the source for pltf to see how to do this.

See Also

imageplt(1), plt(1)

Availability

pltf is available as part of the plt package in PhysioToolkit (see SOURCES below) under the GPL.

Author

George B. Moody (george@mit.edu)

Sources

http://www.physionet.org/physiotools/plt/plt/misc/pltf
http://www.physionet.org/physiotools/plt/plt/misc/ftable.c

Index