GNU graphics

Utilities for plotting scientific data

1 The GNU Graphics Utilities

The GNU graphics utilities are a set of programs for plotting scientific data. The program `graph` reads data files and writes a stream of plotting commands in a device independent format refered to below as a GNU plot file. The remaining programs provide support for displaying GNU plot files on tektronix 4010, PostScript (TM)*, and X window system compatible output devices.

`graph` reads both ascii and binary data files and writes a plot file with or without axes and labels. You can specify labels and ranges for the axes, and you can set the sizes and position of the plot on the page. Each invocation of graph produces a plot with single set of axes and data. You can place an arbitrary number of plots on the page by concatenating the plot output of several invocations.

`plot2ps` is a utility for converting plot files into PostScript. The `plot2ps` utility reads plotting commands from named files or the standard input and writes PostScript to the standard output. You can then print the PostScript output on a printer, or edit it using the `idraw` graphics editor. You can also easily include the output in LaTeX documents using `dvips` and the LaTeX command `psfig`.

Why is this useful? The plot file format is a common standard on un*x systems. To produces figures for publication, you might need to take data sets, and produce labeled figures from them. This can be done using `graph`, `plot2ps` and the `idraw` editor. You can also include these figures in LaTeX documents using the `dvips` utility. All of these utilities, as well as the plot file format and plot library, are discussed in the following sections.

This documentation is under revision. Any comments, suggestions, or additions would greatly benifit GNU users. Please mail them to either '`bug-gnu-utils@prep.ai.mit.edu`' or '`Rich@Rice.edu`'.

* PostScript is a trademark of Adobe Systems Incorporated.

1.1 `graph` Examples

Each invocation of `graph` plots data read from stdin or named files togeather in a single plot with or without axes and labels. The following sections show common usage of `graph`.

1.2 Simple examples using graph

By default, `graph` reads ascii data from the standard input or files specified on the command line. `graph` reads pairs of values, x and y coordinates:

```
0.0  0.0
1.0  0.2
2.0  0.0
3.0  0.4
4.0  0.2
5.0  0.6
```

To plot this data, you might use

```
graph < ascii_data_file |plot
```

where ascii_data_file could contain data similar to the above example. You can replace the command `plot` with `plot2tek` if you have a tektronix 4010 compatible graphics terminal, `plot2ps` if you have a postscript compatible printer or previewer, or `xplot` if you have an X window system display.

Note that `graph` is commonly supplied with some operating systems. If so, some confusion may arise if the system supplied version is executed mistakenly. On unix systems, you can determine which version you invoke by typing the command `which graph`, which prints the file name of the version you invoke by default.

To reduce the change the size of the plot and position it in the middle of the display, you could use

```
graph -h .4 -w .4 -r .2 -u .2 < ascii_data_file |plot
```

where `h` and `w` are the height and width of the plot and `r` and `u` indicate how far up and to the right the plot is positioned.

You can put symbols at each data point using

```
graph -S 2 .01 < ascii_data_file |plot
```

where 2 indicates which symbol to plot, and .01 indicates it's size.

You can choose the type of line draw on each curve:

```
graph -m 2 < ascii_data_file |plot
```

where 2 indicates what kind of line connects the data points.

1.3 The format of input to graph

As mentioned above, by default `graph` reads ascii pairs of values, x and y coordinates, from the standard input or files specified on the command line. Optional labels may be placed after each coordinate in the data file. The label ends at the end of the line:

```
3.0  0.4 this is a label for point (3.0, 0.4).
```

The label must be enclosed in double quotes if it would otherwise be appear to be a coordinate:

```
3.0  0.4 "5.0 looks like a value."
```

You can use the '`-b`' to break lines after each label in the input. Use a pair of empty double quotes if you need to break a line, but do not need a label.

```
0.0  0.0
2.0  0.0
1.0  0.2
""
0.0  0.1
2.0  0.2
4.0  0.3
```

You can also break cuves using '`-M`' option to break lines whenever the abscissal values between successive pairs of points decrease. When using '`-M`', each continuous curve has monotonicly increasing abscissal values.

```
0.0  0.0 first data set
2.0  0.0
4.0  0.2
0.0  0.1 second data set
2.0  0.2
4.0  0.3
```

graph will automaticly generate abscissal values for you if you specify the '-a' option. Only ordinate values are given in the data, and the data is then assumed to be equally sampled along the abscissa. The values following '-a' on the command line specify the sampling interval and the abscissal value of the first data point.

```
0.0
0.1
0.2 label for point (2.0, 0.2)
0.3
0.2
0.3
```

1.4 Combining several data sets in one plot

There are cases where you will want to superimpose several data sets or several plots on top of each other. If for example, the data sets are in seperate files, you can specify each by it's name on the command line. Since graph reads the standard input only if no files are named on the command line, you must add the name -- if you want graph to read the standard input as well.

```
graph data-file-one data-file-two data-file-three |plot
```

For comparison sake, you might wish to distinguish the data in one set from another either by using different symbols at each point or by distinguishing the type of line draw. You can do this by preceeding each file name with options affecting the synbol or line style used to plot that data.

```
graph -S 1 data-file-one -S 3 data-file-two -S 4 data-file-three |plot

or

graph -m 1 data-file-one -m 3 data-file-two -S 4 data-file-three |plot
```

If you need to superimpose several data sets, but must invoke graph seperately for each, you will have to specify the limits of the axes.

```
graph -x 0 100 -y -3 3 -S 3 -m -1 < ascii_data_file_1 >> plot_file
```

where -x 0 100 specifies the limits on the x axis, -y -3 3 specifies the limits on the y axis, -S 2 specifies a box to be drawn at each point, and -m -1 specifies that no line is to be draw connecting the points. You can overlay a second data set on the first by using:

```
graph -s -g 0 -x 0 100 -y -3 3 -m 0 < ascii_data_file_1 >> plot_file
```

where `-s` avoids erasing the page, `-g 0` avoids drawing the axis, tick marks and labels which were drawn previously, and `-m 0` specifies that solid lines are draw connecting the points.

1.5 How to put multiple plots on one page

The command

```
graph -h .4 -w .4 -r .1 -u .1 < ascii_data_file_1 >  plot_file
```

will put a single box containing the plot in the lower left hand quarter of the page. You can add another plot to the upper left hand corner of the page using the command

```
graph -s -h .4 -w .4 -r .1 -u .6 < ascii_data_file_2 >> plot_file
```

Be sure you use the `-s` option so the the first plot isn't erased.

Likewise you can add plots to the right hand side of the page using

```
graph -s -h .4 -w .4 -r .6 -u .1 < ascii_data_file_3 >> plot_file
graph -s -h .4 -w .4 -r .6 -u .6 < ascii_data_file_4 >> plot_file
```

The tick marks can be moved inside the box and labels moved to the opposite sides using

```
graph -T -.005 < ascii_data_file >> plot_file
```

1.6 Reading other data formats

`graph` will read binary data in integer, short integer, float, and double float format when you use the `-d` option followed by `f`, or `d`, respectively. There are two advantage to using binary data: 1) `graph` runs significantly faster because the computational overhead for converting data from ascii to binary is eliminated, and 2) the input files can be significantly smaller than the ascii format would be. Double float is the fastest format to read, while short integer is the most space conservative. If you have very large data sets, using a binary format can reduce storage and runtime costs.

For example, you can create double float data as output from C language programs:

```
#include <stdio.h>
void write_point (x, y)
  double x, y;
{
  fwrite(&x, sizeof (double), 1, stdout);
  fwrite(&y, sizeof (double), 1, stdout);
}
```

You can then plot data written this way using:

```
graph -d d <datafile >plotfile
```

1.7 Graph Command Line Options

The following table describes each of the command line arguments to graph. Each option which takes an argument is followed by the type and default values of the argument in parentheses.

'`-a` [*step_size* [*lower_limit*]]'
'`+auto-abscissa` [*step_size* [*lower_limit*]]'
: (floats, defaults 1 and 0) Automaticly generate abscissa (x) values. This option specifies that the data contains only ordinate (y) values. The difference between successive x values will be *step_size*, and the first x value will be *lower_limit*. To return to reading abscissal values from the input you can specify `-a 0`, which disables automatic generation of the abscissa and returns *step_size* and *lower_limit* to their default values.

'`-b`'
'`+break-on-labels`'
: Assumes multiple data sets are in the data file, and the data sets are separated by a label. The default is don't break on labels.

'`-M`'
'`+break-non-monotone`'
: When successive abscissa (x) values decrease, a separate data set is assumed. This allows multiple data sets in each file. Similar to `-b`.

'`-f` *size*'
'`+fontsize` *size*'
: Specify the size of the desired font as *size* points. Not all display devices will honor this command.

'`-N` *x*`|`*y*'

'`+no-label` *x*`|`*y*'

By default, values at each tick mark are labeled beside the axis. This option removes the labeling of the tick marks on the specified axis.

'`-R`'

'`+dont-round-to-next-tick`'

By default, the ends of the axes are extended to the next tick mark. This option prohibits rounding the limits of the axes to the next tick mark.

'`-c` *string*'

'`+point-label` *string*'

This option defines *string* as the default label for each point. Any label in the input will override this default.

'`-S` *symbol_number* `[`*symbol_size*`]`'

'`+symbol` *symbol_number* `[`*symbol_size*`]`'

(integer and float, defaults -1 and 0.01) Draw a symbol at each point in the data. *symbol_number* specifies the shape of the symbol according to the following table and *symbol_size* specifies the fractional size of the symbol with respect to the height and width of the plot. Note that you can specify symbols to be drawn without any line connecting them by specifying the option `-m -1`.

-1 no symbol at all

0 plus sign (+)

1 cross (x)

2 star (*)

3 box

4 diamond

5 circle

'`-T` *tick_size*'

'`+ticksize` *tick_size*'

(float, default .01) *tick_size* is the fractional size of the tick marks on each axis. A value of 1.0 produces tick marks on the x (y) axis whose length is equal to the width (height) of the plot.

'`-X` *x_label*'

'`+xtitle` *x_label*'

(string, default blank) *x_label* is a label printed below the x axis.

'`-Y` *y_label*'

'`+ytitle` *y_label*'

(string, default blank) *y_label* is a label printed to the right of the y axis.

'`-d` *data-format*'

'`+data-format` *data-format*'

This specifies what format the input data is in. Note labels can be used only in ascii format input files.

'a'
'A' ascii data

'i'
'I' binary integer data

's'
'S' binary short integer data

'f'
'F' binary float data

'd'
'D' binary double data

'+debug' Debugging information, including the data read in, is sent to the standard error output.

'-g *grid_style*'
'+grid *grid_style*'
(integer, default 1) *grid_style* specifies the type of box framing the plot and whether grid lines are drawn inside the box.

0 no box around plot, no axes, no labels.

1 box containing a grid and axes with tick marks and labels.

2 box around plot, tick marks around the box and labels.

3 box around plot, ticks on left and lower sides only and labels.

4 axes intersect at the origin without a box or grid.

'-h *height*'
'+height-plot *height*'
(float, default 0.8) *height* specifies the fractional height of the plot with respect to the height of the plotting area. A value of 1.0 will produce a box which fills the available area. Note that the tick marks and labels are outside this area so that values less than 1.0 are generally used.

'-L *top_label*'
'+toptitle *top_label*'
(string, default blank) *top_label* is a label placed above the plot.

'-m *line_mode*'
'+linestyle *line_mode*'
(integer, default 0) *line_mode* specifies the mode (or style) of lines drawn between data points.

-1 no line at all

0 solid

1 dotted

2 shortdashed

3 dotdashed

4 longdashed

5 disconnected

'`-r` *right*'

'`+right-margin-posn` *right*'

(float, default 0.1) Move the plot to the right by a fractional amount *right* with respect to the width of the plotting area. This produces a margin on the left hand side of the plot. A value of 0.5 will produce a margin half the width of the available area. Note that the tick marks and labels are drawn in the margin.

'`-u` *up*'

'`+bottom-margin-posn` *up*'

(float, default 0.1) Move the plot up by a fractional amount *up* with respect to the height of the plotting area. This produces a margin below the plot. A value of 0.5 will produce a margin half the height of the available area. Note that the tick marks and labels are drawn in the margin.

'`-s`'

'`+save-screen`'

Save the screen. This option prevent graph from erasing the previous contents of the graphics window or device.

'`-t`'

'`+transpose`'

Transpose the abscissa and ordinate. This option causes the axes to be interchanged, and the options which apply to each axis to be applied to the opposite axis. That is, data is read in as (y, x) pairs and '`-x`', '`-X`' and '`-lx`' apply to the y axis.

'`-w` *width*'

'`+width-plot` *width*'

(float, default 0.8) *width* specifies the fractional width of the plot with respect to the width of the plotting area. A value of 1.0 will produce a box which fills the available area. Note that the tick marks and labels are outside this area, so values less than 1.0 are generally used.

'`-x` *lower_limit upper_limit*'

'`+xlimits` *lower_limit upper_limit*'

(floats) The arguments *lower_limit* and *upper_limit* specify the limits of the x axis. By default the upper and lower limits are taken from the data. If unspecified the limits of the data are used.

'`-y` *lower_limit upper_limit*'

'`+ylimits` *lower_limit upper_limit*'

These arguments specify the scale and limits of the y axis as for the x axis above.

'-l *x*|*y*'

'+log-axis *x*|*y*'

The argument indicates which axis should be a log axis. Either one or both x- and y-axes can be specified by using the appropriate letter. Use *xy* or *yx* to specify both.

'+high-byte-first'

'+low-byte-first'

These options force graph to use the specified byte order when writing out the plot file. By default the byte order is host dependent.

1.8 Examples Using `plot2ps`

To produce a plot of data arranged in ordered pairs of x and y coordinates in an ASCII file, you can use:

```
graph <asciiDataFile | plot2ps | lpr -Plw
```

To create a simple PostScript figure you can use:

```
echo 0 0 1 1 2 0 | spline | graph | plot2ps > test.ps
```

To edit the plot:

```
idraw test.ps
```

To use the previewer to look at the plot:

```
gs test.ps
```

1.9 Plot2ps Command Line Options

`plot2ps` is a relatively simple utility in that there are few command line options to choose from. The plot file format does not contain methods for specifying font or font size, so you must specify these things with options. There are no other options for controlling the picture.

The plot file format is machine dependent on the byte order of unformatted, signed, two byte integer coordinates contained in plot commands. The '`-high-byte-first`' or '`-low-byte-first`' option specifies this order explicitly. `plot2ps` attempts to determine the byte order from commands early in the plot file, but the method is heuristic and is not foolproof. Several standard plot sizes specified by the `open` command are used to recognize byte order by `plot2ps`. If these sizes are recognized in byte reversed order, `plot2ps` adjusts accordingly. These sizes include 504x504, 2048x2048 (versatek plotters), 2100x2100, 3120x3120(tektronix 4010 terminals) and 4096x4096 (gsi 300 terminals).

The remaining command line options may be used specify an alternate PostScript prologue and to print the licensing information.

Input plot files names may be specified anywhere on the command line. If no file names are specified, or the name '`-`' is specified, the standard input is read for plotting instructions. Only the font or font size options which precede a file name will affect the text for that file.

'`-H`'
'`+help`' The help option prints a summary of command line syntax for `plot2ps`, a list of the font names (the standard builtin PostScript fonts), and version, copyright and warranty information. Specifying this options causes plot2ps to ignore files on the standard input. You can specify a file on the standard input explicitly with the option '`-`' if you want it to read the standard input as well.

'`-v`'
'`-V`'
'`+version`'
This option prints version, copyright and warranty information.

'`-fontsize` *size*'
'`-f` *size*' The fontsize options specifies the default size in printer's points (1/72 inch) of all text appearing in the plot. If unspecified, the size defaults to 14 points.

Some sizes are supported better than others under X windows. The standard sizes distributed with X windows are 8, 10, 12, 14, 18, and 24 points. Text at these point sizes will display correctly in the `idraw` editor. Other font sizes will print correctly on a PostScript device such as the laserwriter, but may not appear at the correct size in the `idraw` editor.

'`-font` *name*'
'`-fo` *name*'
The font name option specifies the name of the default font for all text appearing in the plot. '`plot2ps -help`' prints a listing of the font names on the standard output. These names include the available builtin fonts on standard PostScript printers.

'-high-byte-first'
'-h' The high-byte-first option specifies explicitly that the higher order byte of each signed, two byte integer occurs first in the file. It disables determination of byte order from the file itself.

'-line-width *width*'
width is the width of lines drawn in the plot, and defaults to a value of 0. A value of 0 will produce the thinest line possible in a device dependent fashion, however this is known to cause problems for older versions of idraw. The line width is device independent for a positive values of *width*.

'-low-byte-first'
'-l' The low-byte-first option specifies explicitly that the lower order byte of each signed, two byte integer occurs first in the file. It disables determination of byte order from the file itself.

'-prologue *filename*'
'-p *filename*'
The prologue option specifies the name of an alternate PostScript prologue *filename* to be used in place of the default `idraw` prologue. The prologue declares procedures used to draw each graphic object. The default prologue was generously provided by John Interante and is a part of the InterViews distribution, version 2.5.

'-copying'
'-warranty'
The copying and warranty options print a copy of the GNU General Public License on the standard error output. Included is conditions for copying `plot2ps` and information on the lack of any warranty.

These conditions do not cover the output of `plot2ps`. The only conditions imposed on the output are those which come from the prologue that you are using.

'-signed'
'-unsigned'
The signed and unsigned options specify whether coordinates in the plot file are signed. By convention, coordinates are always signed. Some plot files do not follow this convention, and you can use the unsigned option to convert those files.

'-bbox' The bbox option specifies that a bounding box comment will be written at the end of the output file. This information is useful for document preparation systems which determine how to size and place the figure using the bounding box. See also the atend script.

1.10 A plot file previewer for X11

`xplot` is a plot file previewer for the X window system. It reads GNU plot commands from its standard input and draws the resulting graphics in an X window.

After xplot reaches the end-of-file on the input, it puts itself in the background (forks). Control returns to the calling program, while xplot continues, remaining on screen.

To exit, click the left mouse button in the xplot window. Note that xplot ignores SIGHUP signals, so you must use another signal to kill xplot if necessary.

1.11 Xplot Options

`xplot` accepts all of the standard X toolkit command line options, and the initial geometry specification determines the resolution, with a default geometry of 500x500 pixels.

The following standard X Toolkit command line arguments may be used with `xplot`:

'`-bg` *color*'
: This specifies the color to use for the background of the window. The default is *white*.

'`-bd` *color*'
: This specifies the color to use for the border of the window. The default is *black*.

'`-bw` *number*'
: This specifies the width in pixels of the border surrounding the window.

'`-fg` *color*'
: This specifies the color to use for displaying text. The default is *black*.

'`-fn` *font*'
: This specifies the font to be used for displaying normal text. The default is *6x10*.

'`-rv`'
: This indicates that reverse video should be simulated by swapping the foreground and background colors.

'`-geometry` *geometry*'
: This specifies the preferred size and position of the plot window.

'`-display` *host*:*display*'
: This specifies the X server to contact.

'`-xrm` *resourcestring*'
: This specifies a toolkit resource property. See the manual page for xrdb.

`xplot` uses the athena Command widget in the X Toolkit. So, it understands all of the core resource names and classes as well as:

'reverseVideo'
: (class ReverseVideo) Specifies that the foreground and background colors should be reversed.

1.11.1 Example

The resources:

```
Xplot.font: 6x9
Xplot.geometry: 300x300
```

will set the font used in the plot window to '6x9' and the size of the window to 300 by 300 pixels.

1.12 Using Plot2fig

To create a simple plot file one can use:

```
echo 0 0 1 1 2 0 | spline | graph | plot2fig > test.fig
```

To edit the plot:

```
fig test.fig
```

To convert the fig file into dvi code, create a latex file containing a document which includes the figure:

```
\\documentstyle\[\]{article}
\\begin{document}
\\input{test}
\\end{document}
```

Then, run transfig on the figure and latex on the document:

```
% transfig -L latex test.fig
% make test.tex
% latex t.tex
```

To edit a plot of data arranged in ordered pairs of x and y coordinates in an ascii file, one can use:

```
% graph <asciiDataFile | plot2fig >file.fig
% fig file.fig
```

1.13 A plot file to fig file translator

`plot2fig` reads plotting instructions from the specified input files and/or the standard input and produces `Fig` compatible code on it's standard output. This output file can be edited with the fig (Facility for Interactive Generation of figures) graphics editor. The output can subsequently be converted to pictex, PostScript, latex, epic, eepic, and tpic languages using the `transfig` translator.

Any unrecognized options on the command line are assumed to be input files. The standard input is read by default only if no other files specified on the command line are successfully opened. A single dash (-) on the command line indicates the standard input is to be read. Each option is set and each file read in the order they are specified on the command line.

For compatibility with pic2fig, plot2fig ignores leading white space in labels. Labels containing all white space are ignored.

'`-fn` *name*'
'`-fontname` *name*'
: Default: the default font of the transfig output device. This option sets the font for all subsequent text to *name*. Recognized font names are typewriter, modern, italic, bold, and times. In addition, courier is an alias for typewriter and roman is an alias for times. Note that the fonts are device dependent.

'`-fs` *size*'
'`-fontsize` *size*'
: Default: 12. This option sets the size of subsequent text to *size* (in printer's points).

'`-h`'
'`-high-byte-first`'
: This option specifies that the byte ordering of two byte integers in the input plot file is high byte first.

'`-l`'

'`-low-byte-first`'

This option specifies that the byte ordering of two byte integers in the input plot file is low byte first.

'`-warranty`'

'`-copying`'

This option prints out the copying conditions and warranty information.

'`-signed`'

'`-unsigned`'

Default: signed. This option specifies whether two byte integers in the input plot file are unsigned or signed.

'`-`' This option specifies explicitly that the standard input should be read for plotting instructions.

1.14 Tektronix 4010 output

'`-H`'

'`+help`' The help option prints a summary of command line syntax, a list of the known font names, and version, copyright and warranty information. Specifying this options causes plot to ignore the standard input, so you must specify the option '`-`' if you want it to read the standard input as well.

'`-v`'

'`-V`'

'`+version`'

This option prints version, copyright and warranty information.

'`-f` *size*'

'`+fontsize` *size*'

The fontsize options specifies the default size in printer's points (1/72 inch) of all text appearing in the plot. If unspecified, the size defaults to 14 points.

'`+high-byte-first`'

'`+low-byte-first`'

These options force graph to use the specified byte order when writing out the plot file. By default the byte order is host dependent.

'`-F` *name*'

'`+fontname` *name*'

The font name option specifies the name of the default font for all text appearing in the plot. '`plot -help`' prints a listing of the font names on the standard output. These names include the available builtin fonts on standard PostScript printers.

'`-h`'

'`+high-byte-first`'
: The high-byte-first option specifies explicitly that the higher order byte of each signed, two byte integer occurs first in the file. It disables determination of byte order from the file itself.

'`-l`'

'`+low-byte-first`'
: The low-byte-first option specifies explicitly that the lower order byte of each signed, two byte integer occurs first in the file. It disables determination of byte order from the file itself.

'`-p` *filename*'

'`+prologue` *filename*'
: The prologue option specifies the name of an alternate PostScript prologue *filename* to be used in place of the default `idraw` prologue. The prologue declares procedures used to draw each graphic object. The default prologue was generously provided by John Interante and is a part of the InterViews distribution, version 2.5.

'`+copying`'

'`+C`'

'`+warranty`'

'`+W`'
: The copying and warranty options print a copy of the GNU General Public License on the standard error output. Included is conditions for copying `plot` and information on the lack of any warranty.

'`-s`'

'`+signed-input`'

'`-u`'

'`+unsigned-input`'
: The signed and unsigned options specify whether coordinates in the plot file are signed. By convention, coordinates are always signed. Some plot files do not follow this convention, and you can use the unsigned option to convert those files.

1.15 Including a Figure in an Article

This is an example of LaTeX code which places the figure generated in the previous example in a page of text.

```
\documentstyle[]{article}
\input{psfig}
\begin{document}
\title{Title of the article.}
```

```
\author{The Author's name}
\maketitle
This is an example of how to include PostScript figures in LaTeX documents.
\begin{figure}[h]
\centerline{\psfig{figure=test.ps,height=3in}}
\caption{Here is a description of the figure which will appear below it.}
\end{figure}
Note that the above figure was included using dvips.
\end{document}
```

If the above LaTeX code is contained in a file called '`mytext.tex`' you can use the commands

```
latex mytext
dvips mytext.dvi >mytext.ps
lpr -Plw mytext.ps
```

to format and print the example text.

1.16 Options to `psfig` for Including Figures

`psfig` is a LaTeX command used to insert a PostScript figure into a document.

`psfig` can be used to insert `plot2ps` generated PostScript into a LaTeX document. The placement of the `psfig` command tells LaTeX where in the document to place the PostScript, and arguments to the command give the name of the file containing the PostScript, and the desired size of the figure. Arguments are separated by commas or blanks, and are of the form '*keyword*=*value*'. The following is a list of valid arguments for the `psfig` command:

`file=`*name*
: The file name of the PostScript figure.

`height=`*size*
: The height of the figure (eg. 3in). If you specify only a height or only a width, the width and height are scaled equally. If you specify both a width and a height the aspect ratio will be affected.

`width=`*size*
: The width of the figure (eg. 3in).

`bbllx=`*coordinate*
: The bounding box lower left-hand x coordinate. Any PostScript file which conforms to the PostScript Document Structuring Conventions version 2.0 should contain a bound-

ing box information at the head of the file. `plot2ps` output conforms to the version 2.0 conventions so that you should not need to use any of the bounding box options.

`bblly=`*coordinate*
: The bounding box lower left-hand y coordinate.

`bburx=`*coordinate*
: The bounding box upper right-hand x coordinate.

`bbury=`*coordinate*
: The bounding box upper right-hand y coordinate.

`rheight=`*size*
: Horizontal space to reserve for the figure.

`rwidth=`*size*
: Vertical space to reserve for the figure.

`clip=`
: Clip the figure. `clip=` is a switch and takes no value, but the '`=`' must be present. This option is useful for including PostScript figures which use the size of the clipping path to size themselves.

1.17 A perl script for use with psfig

`atend.pl` is a perl script which moves the bounding box comment from the trailer to the header. Although either is legal, most document preparation software, such as `psfig`, will only accept bounding box comments in the header. If you use `psfig` and the `-bbox` option togeather, run `atend.pl` on the output of `plot2ps` before importing the graphics using `psfig`.

`atend.pl` can be used as a filter:

```
echo 0 0 1 1 2 0 | spline | graph >spline.pl
plot2ps -bbox |atend.pl - >spline.ps
```

Or, `atend.pl` can be used to fix the output file in place:

```
echo 0 0 1 1 2 0 | spline | graph |plot2ps -bbox >spline.ps
atend.pl spline.ps
```

1.18 How to Get `dvips`

The `dvips` utility mentioned previously is used convert dvi files generated by LaTeX into postscript. It also has support for inclusion of PostScript figures into LaTeX documents. It is available via anonymous ftp from '`labrea.stanford.edu`' (36.8.0.47). Look for '`pub/dvips.tar.Z`'.

1.19 How to Get `idraw`

The `idraw` utility mentioned previously is an interactive graphics editor which is distributed with InterViews. InterViews is available via anonymous ftp from '`interviews.stanford.edu`' (36.22.0.175) in the file '`InterViews/2.5.tar.Z`'.

1.20 How to Get `ghostscript`

Ghostscript is a previewer which is intended to be compatible with the PostScript language. It supports several output devices including the X window system and ega displays. Version 2.0 is suitable for previewing LaTeX documents with imbedded encapsulated PostScript figures, the type of PostScript figures generated by plot2ps and idraw. It is available via anonymous ftp from '`prep.ai.mit.edu`' (18.71.0.38) - Look for '`pub/gnu/ghostscript.tar.Z`'.

1.21 `libps`, a Library of Plot Functions

Libps is a library of plot functions for drawing graphic object using PostScript. Before drawing any objects or using any of the other functions, a program should call `openpl`. Before exiting and after all other libps calls a program should call `closepl`.

1.21.1 `alabel`

int `alabel` (char *x_justify*, char *y_justify*, char **label*);

`alabel` takes three arguments *x_justify*, *y_justify*, and *label* and places the label according to the x and y axis adjustments specified in *x_justify* and *y_justify* respectively. *x_justify* is a character containing either '`l`', '`c`', or '`r`' for left, center or right justified with respect to the current x coordinate. *y_justify* is a character containing either '`b`', '`c`', or '`t`' for placing the bottom center or top of the label even with the current y coordinate. **label* is a string containing the label. The current point is moved to follow the end of the text.

See Section 1.21.9 [fontname], page 22 on how to change the default font. See Section 1.21.10 [fontsize], page 23 on how to change the font size.

1.21.2 `arc`

int `arc` (int *x*, int *y*, int *x0*, int *y0*, int *x1*, int *y1*)

`arc` takes six integer arguments specifying the coordinates of the center (*x*, *y*), beginning (*x0*, *y0*), and ending (*x1*, *y1*) of a circular arc. The current point becomes (*x*, *y*).

1.21.3 `circle`

int `circle` (int *x*, int *y*, int *r*)

`circle` takes three integer arguments specifying the center (*x*, *y*) of the circle and its radius (*r*). The current point becomes (*x*, *y*).

1.21.4 `closepl`

int `closepl` ()

`closepl` takes no arguments. It merely outputs the PostScript trailer containing a `showpage` command.

1.21.5 `color`

int `color` (int *red*, int *green*, int *blue*);

`color` sets the foreground color of all the following objects. The arguments *red*, *green* and *blue* indicate the intensity of red, green and blue components of the foreground color respectively. Each is a unsigned integer specifying an intensity in the range from 0 to 0xFFFF. A value of (0, 0, 0) represents black and a value of (0xFFFF, 0xFFFF, 0xFFFF) indicates white.

1.21.6 `cont`

int `cont` (int x, int y)

`cont` takes two integer arguments specifying the coordinate (x, y) for the continuation of a line. This draws a line segment from the current point to the point (x, y). The current point then becomes (x, y).

1.21.7 `erase`

int `erase` ()

`erase` normally erases all the graphics from the display before a plot is viewed. Since we start off with a blank page in PostScript and `idraw` this function does nothing.

1.21.8 `fill`

int `fill` (int *level*);

`fill` sets the intensity of the filler for closed paths. The argument *level* indicates the grey level of the fill pattern. It's value ranges from 1 to 0xFFFF. A value of 1 represents black and a value of 0xFFFF indicates white. A value of 0 represents no fill, or transparent.

1.21.9 `fontname`

int `fontname` (char **font_name*);

`fontname` takes a single string argument, *font_name*, specifying the name of the font to be used for following text. The laser writer builtin fonts are supported:

```
courier-bold
courier-boldoblique
courier-oblique
courier
helvetica-bold
helvetica-boldoblique
```

```
helvetica-oblique
helvetica
symbol
times-bold
times-bolditalic
times-italic
times-roman
```

1.21.10 `fontsize`

int `fontsize` (int *size*);

`fontsize` takes a single integer argument *size* in printer's points (1/72 inch) and sets the font size accordingly.

1.21.11 `label`

int `label` (char **s*)

`label` takes a single string argument *s* and draws the text contained in *s* at the most recently used coordinate in the current font. By default the text is left justified and centered vertically with respect to the current coordinate.

1.21.12 `line`

int `line` (int *x1*, int y*1*, int *x2*, int *y2*)

`line` takes four integer arguments specifying the beginning (*x1*, *y1*) and ending (*x2*, *y2*) points of a line. The current point becomes (*x2*, *y2*).

See Section 1.21.13 [linemod], page 23 for how to specify the style or pattern of line.

1.21.13 `linemod`

int `linemod` (char **s*)

`linemod` takes a single string argument *s* containing the name of the line style desired. The names supported are longdashed, disconnected, dotdashed, dotted, solid and shortdashed. These correspond to the following sixteen bit patterns:

```
solid            --------------------------------
longdashed       -------         -------
disconnected     -               -
dotdashed        -----------  -  -----------  -
dotted           - - - - - - - - - - - - - - - -
shortdashed      --              --
```

1.21.14 move

int **move** (int *x*, int *y*)

`move` takes two integer arguments specifying the coordinate (*x*, *y*) for the beginning of a new line. This is equivalent to lifting the pen on a plotter and moving it to a new position without drawing any line. The current point becomes (*x*, *y*).

1.21.15 openpl

int **openpl** ()

`openpl` normally opens the device. For PostScript we just print out the PostScript prologue. The following global variables defined in `openpl` specify what prologue is written to the output.

user_has_prologue is a flag. If it is non-zero then the open routine should output the user specified prologue contained in the file specified in the string *users_prologue*.

users_prologue is a string containing the file name for any user specified PostScript prologue. This file is a substitute for the default prologue.

1.21.16 point

int **point** (int *x*, int *y*)

`point` takes a pair of integer arguments specifying the coordinate (x, y) for a single point. The current point then becomes (x, y).

1.21.17 `rotate`

int `rotate` (int *angle*);

`rotate` takes three integer arguments. The last argument, *angle*, specifies the angle in degrees counter-clockwise from the x (horizontal) axis following text.

1.21.18 `space`

int `space` (int *x0*, int *y0*, int *x1*, int *y1*)

`space` takes two pair of integers arguments specifying the lower, left-hand and upper, right-hand limits of the range of plot coordinates. The scaling of input to output coordinate conversion is adjusted to fit these ranges into the page. Note however that if the ranges of x and y coordinates are different the smallest scaling of the two is used to avoid affecting the aspect ratio of the plot. This means that although the plot is scaled to fit on the page, the axes are not stretched with respect to each other.

1.22 The Plot File Format

The plot file is a set of plotting commands and data. Each command is a single ascii character indicating which operation is to be performed. The data following a command is either a newline terminated ascii string or several signed, two byte integers in binary format. For example, the command to move the current point to the coordinate (3,5) would be `m\000\003\000\005`.

Note that the byte order of the binary representation of the signed, two byte integers is machine dependent, so on some machines, this command might appear as `m\003\000\005\000`. `plot2ps` tries to guess the byte order from the arguments to the `openpl` command and adjust the order accordingly.

The following table lists each single character commands followed by the name of the corresponding libps function called to handle the data and a description of the command and data.

`Command` Description

`a` The arc command is followed by three pair of signed, two byte integers indicating the center, starting and ending points for a circular arc. The center becomes the the current point. This is equivalent to the `arc` function (see Section 1.21.2 [arc], page 21).

`c` The circle command is followed by three signed, two byte integers. The first two indicate the x and y coordinates of the center of the circle and the third indicates the radius of the circle. The center becomes the the current point. This is equivalent to the `circle` function (see Section 1.21.3 [circle], page 21).

`C` The color command is followed by three unsigned, two byte integer which indicate the intensity of *red*, *green* and *blue* components respectively of the background color. For each component the range of intensity is from 0 to 65535. A value of (*0*, *0*, *0*) represents black and (*65535*, *65535*, *65535*) represents white. This is equivalent to the `color` function (see Section 1.21.5 [color], page 21).

`e` The erase command is followed by no data. The erase command is not needed since in `idraw` and PostScript we start off with a blank page. For this reason the erase command does not actually output any PostScript. This is equivalent to the `erase` function (see Section 1.21.7 [erase], page 22).

`f` The linemod command is followed by a newline terminated string containing the name of the line mode (or style) for all subsequent lines, circles and arcs. This is equivalent to the `linemod` function (see Section 1.21.13 [linemod], page 23) which describes the line styles and their names.

`F` The the fontname command is followed by a newline terminated string containing the name of the font to be used for all subsequent text. This is equivalent to the `fontname` function (see Section 1.21.9 [fontname], page 22).

`l` The line command is followed by two pair of signed, two byte integers which indicate the starting and ending points of the line. The second pair becomes the the current point. This is equivalent to the `line` function (see Section 1.21.12 [line], page 23).

`L` The fill command is followed by an unsigned, two byte integer indicating the intensity of the fill for closed paths. A value of 1 represents black and a value of 0xFFFF indicates white. The value 0 is special in that is indicates that no solid fill should occur, and that the interior of the respective path is transparent. This is equivalent to the `fill` function (see Section 1.21.8 [fill], page 22).

`m` The move command is followed by a pair of signed, two byte integers containing the location of the new current point. No line is drawn to this point as opposed to the continue command (`c`) which draws a line. This is equivalent to the `move` function (see Section 1.21.14 [move], page 24).

`n` The continue command is followed by pair of signed, two byte integers containing the coordinates of the endpoint of a line segment. A line is drawn from the previous current point if it was set using a command such as move or continue. This then becomes the

the current point. This is equivalent to the `cont` function (see Section 1.21.6 [cont], page 22).

‘p’
: The point command is followed by pair of signed, two byte integers containing the location of single point to be drawn. This then becomes the the current point. This is equivalent to the `point` function (see Section 1.21.16 [point], page 24).

‘r’
: The rotate command is followed by one signed, two byte integer. It indicates the rotation of all subsequent text. The rotation is in degrees counter-clockwise from the x (horizontal) axis. This is equivalent to the `rotate` function (see Section 1.21.17 [rotate], page 25).

‘s’
: The space command is followed by two pair of signed, two byte integers which indicate the the lower right-hand and upper left-hand corners of the range of plot coordinate space. `plot2ps` uses the third signed, two byte integer (the right-hand limit) to try to determine the byte order. This is equivalent to the `space` function (see Section 1.21.18 [space], page 25) which describes the recognized sizes.

‘S’
: The fontsize command is followed by an signed, two byte integer containing the size in printers points of all subsequent text. This is equivalent to the `fontsize` function (see Section 1.21.10 [fontsize], page 23).

‘t’
: The label command is followed by a newline terminated string contains a label which is printed at the current point. It is left justified and centered vertically with respect to the current point. The current point is then set at the end of the text. This is equivalent to the `label` function (see Section 1.21.11 [label], page 23).

‘T’
: The adjusted label command is followed by two characters which indicate the horizontal and vertical justification respectively and a newline terminated string containing the label. The label is drawn with the specified justification and the current point is set at the end of the text. This is equivalent to the `alabel` function (see Section 1.21.1 [alabel], page 20) which describes how to specify justification.

1.23 Acknowledgements

Rich Murphey `<Rich@Rice.edu>` wrote the first version of the graph, plot2tek, plot2ps and tek2plot and the documentation. Richard Stallman `<Rms@ai.mit.edu>` further directed development of the programs and editorial support for the documentation. John Interrante generously provided the PostScript prologue and helpful comments on the program.

Arthur Smith (Lassp, Cornell University) `<arthur@helios.tn.cornell.edu>` has generously provided his code for the xplot utility.

Ray Toy `<toy@dino.ecse.rpi.edu>` provided code for graph and tek 4011 and rewrote the tick

mark spacing code, incorporated gnu getopt and provided the statistics package.

David B. Rosen <rosen@bu.edu>, jeffrey templon <templon@copper.ucs.indiana.edu> and David W. Forslund <dwf%hope.ACL@lanl.gov> tested alpha versions.

1.24 Function Index

A

`alabel` (char *x'justify*, char *y'justify*, char **label*) .. 20

`arc` (int *x*, int *y*, int *x0*, int *y0*, int *x1*, int *y1*) .. 21

C

`circle` (int *x*, int *y*, int *r*) 21

`closepl` () 21

`color` (int *red*, int *green*, int *blue*) 21

`cont` (int *x*, int *y*) 22

E

`erase()` 22

F

`fill` (int *level*) 22

`fontname` (char **font'name*) 22

`fontsize` (int *size*) 23

L

`label` (char **s*) 23

`line` (int *x1*, int y*1*, int *x2*, int *y2*) 23

`linemod` (char **s*) 23

M

`move` (int *x*, int *y*) 24

O

`openpl()` 24

P

`point` (int *x*, int *y*) 24

R

`rotate` (int *w*, int *h*, int *angle*) 25

S

`space` (int *x0*, int *y0*, int *x1*, int *y1*) 25

Table of Contents

1 The GNU Graphics Utilities **1**

1.1 `graph` Examples 1
1.2 Simple examples using graph 2
1.3 The format of input to `graph` 3
1.4 Combining several data sets in one plot 4
1.5 How to put multiple plots on one page 5
1.6 Reading other data formats 5
1.7 Graph Command Line Options 6
1.8 Examples Using `plot2ps` 10
1.9 Plot2ps Command Line Options 10
1.10 A plot file previewer for X11 13
1.11 Xplot Options 13
1.11.1 Example 14
1.12 Using Plot2fig 14
1.13 A plot file to fig file translator 15
1.14 Tektronix 4010 output 16
1.15 Including a Figure in an Article 17
1.16 Options to `psfig` for Including Figures 18
1.17 A perl script for use with psfig 19
1.18 How to Get `dvips` 19
1.19 How to Get `idraw` 20
1.20 How to Get `ghostscript` 20
1.21 `libps`, a Library of Plot Functions 20
1.21.1 `alabel` 20
1.21.2 `arc` 21
1.21.3 `circle` 21
1.21.4 `closepl` 21
1.21.5 `color` 21
1.21.6 `cont` 22
1.21.7 `erase` 22
1.21.8 `fill` 22
1.21.9 `fontname` 22
1.21.10 `fontsize` 23
1.21.11 `label` 23
1.21.12 `line` 23
1.21.13 `linemod` 23
1.21.14 `move` 24
1.21.15 `openpl` 24

1.21.16 `point` 24
1.21.17 `rotate` 25
1.21.18 `space` 25
1.22 The Plot File Format 25
1.23 Acknowledgements 27
1.24 Function Index 28