GNUPLOT

An Interactive Plotting Program

Thomas Williams & Colin Kelley
Version 4.1 organized by: Hans-Bernhard Broeker

Mailing list for comments: gnuplot-info@lists.sourceforge.net
Mailing list for bug reports: gnuplot-bugs@lists.sourceforge.net

This manual was prepared by Dick Crawford
3 December 1998

Major contributors (alphabetic order):

- Hans-Bernhard Broeker
- John Campbell
- Robert Cunningham
- David Denholm
- Gershon Elber
- Roger Fearick
- Carsten Grammes
- Lucas Hart
- Lars Hecking
- Thomas Koenig
- David Kotz
- Ed Kubaitis
- Russell Lang
- Alexander Lehmann
- Alexander Mai
- Ethan A Merritt
- Petr Mikulik
- Carsten Steger
- Tom Tkacik
- Jos Van der Woude
- Alex Woo
- James R. Van Zandt
- Johannes Zellner

1 gnuplot

1.1 Copyright


```
AUTHORS


Original Software:
   Thomas Williams,  Colin Kelley.


Gnuplot 2.0 additions:
   Russell Lang, Dave Kotz, John Campbell.


Gnuplot 3.0 additions:
   Gershon Elber and many others.


Gnuplot 4.0 additions:
   See list of contributors at head of this document.
```

1.2 Introduction

'gnuplot' is a command-driven interactive function and data plotting program. It is case sensitive (commands and function names written in lowercase are not the same as those written in CAPS). All command names may be abbreviated as long as the abbreviation is not ambiguous. Any number of commands may appear on a line (with the exception that 'load' or Section 2.2 [call], page 35 must be the final command), separated by semicolons (;). Strings are indicated with quotes. They may be either single or double quotation marks, e.g.,

```
load "filename"
cd 'dir'
```

although there are some subtle differences (see 'syntax' for more details).

Any command-line arguments are assumed to be names of files containing 'gnuplot' commands, with the exception of standard X11 arguments, which are processed first. Each file is loaded with the 'load' command, in the order specified. 'gnuplot' exits after the last file is processed. When no load files are named, 'gnuplot' enters into an interactive mode. The special filename "-" is used to denote standard input. See help for 'batch/interactive' for more details.

Many 'gnuplot' commands have multiple options. Version 4 is less sensitive to the order of these options than earlier versions, but some order-dependence remains. If you see error messages about unrecognized options, please try again using the exact order listed in the documentation.

Commands may extend over several input lines by ending each line but the last with a backslash (\). The backslash must be the _last_ character on each line. The effect is as if the backslash and newline were not there. That is, no white space is implied, nor is a comment terminated. Therefore, commenting out a continued line comments out the entire command (see 'comments'). But note that if an error occurs somewhere on a multi-line command, the parser may not be able to locate precisely where the error is and in that case will not necessarily point to the correct line.

In this document, curly braces ({}) denote optional arguments and a vertical bar (|) separates mutually exclusive choices. 'gnuplot' keywords or Section 2.6 [help], page 45 topics are indicated by backquotes or 'boldface' (where available). Angle brackets (<>) are used to mark replaceable tokens. In many cases, a default value of the token will be taken for optional arguments if the token is omitted, but these cases are not always denoted with braces around the angle brackets.

For on-line help on any topic, type Section 2.6 [help], page 45 followed by the name of the topic or just Section 2.6 [help], page 45 or '¿ to get a menu of available topics.

The new 'gnuplot' user should begin by reading about 'plotting' (if on-line, type 'help plotting').

See the simple.dem demo, also available together with other demos on the web page http://www.gnuplot.info/demo/simple.html

1.3 Seeking-assistance

There is a mailing list for 'gnuplot' users. Note, however, that the newsgroup

```
comp.graphics.apps.gnuplot
```

is identical to the mailing list (they both carry the same set of messages). We prefer that you read the messages through the newsgroup rather than subscribing to the mailing list. Instructions for subscribing to gnuplot mailing lists may be found via the gnuplot development website on SourceForge http://sourceforge.net/projects/gnuplot

The address for mailing to list members is:

```
gnuplot-info@lists.sourceforge.net
```

Bug reports and code contributions should be mailed to:

```
gnuplot-bugs@lists.sourceforge.net
```

The list of those interested in beta-test versions is:

```
gnuplot-beta@lists.sourceforge.net
```

There is also the canonical (if occasionally out-of-date) gnuplot web page at

http://www.gnuplot.info

Before seeking help, please check the

FAQ (Frequently Asked Questions) list.

When posting a question, please include full details of the version of 'gnuplot', the machine, and operating system you are using. A _small_ script demonstrating the problem may be useful. Function plots are preferable to datafile plots. If email-ing to gnuplot-info, please state whether or not you are subscribed to the list, so that users who use news will know to email a reply to you. There is a form for such postings on the WWW site.

1.4 New Features

Gnuplot version 4.1 contained many features introduced since the preceding official version 4.0. This section lists major additions and gives a partial list of changes and minor new features.

1.4.1 New plot styles

1.4.1.1 Histogram

See 'histograms'.

1.4.1.2 Label plots

See 'labels'.

1.4.1.3 Image data mapped onto plot

See 'image'.

1.4.1.4 Filled curves

See 'filledcurves'.

1.4.1.5 Vectors

See ‘vectors‘.

1.4.2 String handling

1.4.2.1 String and text data read from datafiles

See ‘datastrings‘

1.4.2.2 User-defined string variables, operators, and functions

See Section 2.21.72 [variables], page 149

1.4.3 Auto-layout of multiple plots on a page

See Section 2.21.38 [multiplot], page 111.

1.4.4 Input from binary data files

See ‘binary‘.

1.4.5 New or revised terminal drivers

1.4.5.1 Consolidated gif/jpeg/png driver

1.4.5.2 New Enhanced MetaFile (EMF) driver

1.4.5.3 Adobe Illustrator (ai) driver deprecated

1.4.5.4 epslatex, pslatex support consolidated into the main postscript driver

1.5 Features introduced in version 4.0

Gnuplot version 4.0 contained many features introduced since the preceding official version 3.7. These are summarized here.

1.5.1 Mouse and hotkey support in interactive terminals

Interaction with the current plot via mouse and hotkeys is supported for the X11, OS/2 Presentation Manager, ggi and Windows terminals. See ‘mouse input‘ for more information on mousing. See help for Section 1.15.1 [bind], page 26 for information on hotkeys. Also see the documentation for individual mousing terminals ‘ggi‘, ‘pm‘, ‘windows‘ and ‘x11‘.

Sample script: mousevariables.dem

1.5.2 New terminals

‘aqua‘: New terminal for Mac OS X. Requires AquaTerm 1.0 or later.

‘epslatex‘: New terminal. Prepares eps figures for inclusion in LaTeX documents.

‘gif‘: Consolidated with png/jpeg terminals. Requires libgd.

‘ggi‘: New full-screen interactive terminal for Linux. Interface to the General Graphics Interface Library.

‘pdf‘: New terminal exporting Adobe Portable Document Format. Requires libpdf.

‘png‘ and ‘jpeg‘: Support for GIF, PNG and JPEG image output is provided by a new driver via libgd. The new driver supports many more features than the old png driver, including TrueType fonts. Requires libgd.

‘svg‘: New terminal exporting Scalable Vector Graphics.

1.5.3 New plot style pm3d

The ‘splot‘ command is now capable of plotting 2D maps and 3D surfaces colored by greyscale or color palettes. See help for pm3d, Section 2.21.50 [palette], page 120, Section 2.21.118 [cbrange], page 162, ‘set view map‘, ‘set colorbox‘ and Section 2.21.50 [palette], page 120.

Sample scripts: pm3d.dem pm3dcolors.dem pm3dgamma.dem

1.5.4 Filled boxes

A solid color or patterned fill style can be set for any plot style that contains boxes. See ‘boxes‘, ‘boxerrorbars‘, ‘boxxyerrorbars‘, ‘candlesticks‘, ‘set style fill‘.

Sample scripts: fillstyle.dem candlesticks.dem

1.5.5 New plot option smooth frequency

Input data can be filtered through several built-in routines for interpolation or approximation of data. See Section 2.12.1.6 [smooth], page 57, ‘frequency‘, ‘unique‘.

Sample scripts: step.dem mgr.dem

1.5.6 Improved text options

Most gnuplot plot commands that produce text labels now accept modifiers to specify text color, font, size, and rotation angle. See Section 2.21.29 [label], page 104. Not all terminal types support these options, however. The enhanced text mode previously available for the postscript and pm terminals has been extended to other terminal types as well. Terminal types currently supported include aqua, dumb, jpeg, pdf, pm, png, postscript, and x11. See ‘enhanced text‘.

Sample scripts: textcolor.dem textrotate.dem

1.5.7 More text encodings

Several terminals, including Section 2.21.50.6 [postscript], page 126, ‘x11‘ and ‘pm‘, support additional text ‘encodings‘: ISO 8859-1 (Latin 1), ISO 8859-2 (Latin 2), ISO 8859-15 (variant of 8859-1 with Euro sign), KOI8-R and KOI8-U (cyrillic), and miscellaneous codepages. See Section 2.21.18 [encoding], page 91 for more details.

1.5.8 Arrows

Single- or double-ended arrows can be placed on a plot individually from the command line or from a data file via the ‘plot with vectors‘ style. See Section 2.21.2 [arrow], page 75, ‘plotting styles vectors‘.

Sample scripts: arrowstyle.dem vector.dem

1.5.9 Data file format

The new Section 2.21.14 [datafile], page 86 command can be used to specify information about the format of input data files, including the characters used to separate fields, to indicate comment lines, and to specify missing data. Gnuplot now attempts to recognize text fields with embedded blanks as single entities based on the datafile format settings. This allows input from csv (comma-separated value) files such as those exported by spreadsheet programs. See Section 2.21.14 [datafile], page 86. See also the 'binary' option in version 4.1.

1.5.10 New commands

'set view map' selects a top-view 2D projection of 3D surface plot.

'set term push' and 'set term pop' save and restore the current terminal type.

'load' and Section 2.20 [save], page 73 commands accept piped input and output, respectively.

1.5.11 Other changes and additions

Since gnuplot 4.0, 'unset <something>' is preferred to 'set no<something>'. The older form has been deprecated. Version 4.1 continues to allow the older syntax, but such backwards compatibility may be lost in future versions.

Commands of the form 'set <something> <style>' also are deprecated in favor of the more general form 'set style <something> <options>'. Many more plot elements now have style options of their own, including arrows, filled areas, lines, and points. There are also style settings for input data and formatting. Please see Section 2.21.58 [style], page 130, Section 2.21.15 [decimalsign], page 89, and Section 2.21.14 [datafile], page 86.

The MS Windows package includes an additional executable 'pgnuplot.exe' to support piping command through standard input, which is otherwise not available for graphical applications on this system.

1.5.12 Accompanying documentation

In directory docs/psdocs/ you may find new information in the gnuplot output postscript file guide, list of postscript symbols in different encodings.

Improved FAQ. Please read it before asking your question in a public forum.

There are plenty of new demos *.dem in the demo/ directory. Please run them, for example by

```
load "all.dem"
```

before asking for help. Plots produced by the demo scripts can also be viewed at http://www.gnuplot.info/demo/

1.6 Batch/Interactive Operation

'gnuplot' may be executed in either batch or interactive modes, and the two may even be mixed together on many systems.

Any command-line arguments are assumed to be names of files containing 'gnuplot' commands (with the exception of standard X11 arguments, which are processed first).

Each file is loaded with the ‘load‘ command, in the order specified. ‘gnuplot‘ exits after the last file is processed. When no load files are named, ‘gnuplot‘ enters into an interactive mode. The special filename "-" is used to denote standard input.

Both the Section 2.4 [exit], page 36 and Section 2.15 [quit], page 71 commands terminate the current command file and ‘load‘ the next one, until all have been processed.

Examples:

To launch an interactive session:

```
gnuplot
```

To launch a batch session using two command files "input1" and "input2":

```
gnuplot input1 input2
```

To launch an interactive session after an initialization file "header" and followed by another command file "trailer":

```
gnuplot header - trailer
```

1.7 Command-line-editing

Command-line editing is supported by the Unix, Atari, VMS, MS-DOS and OS/2 versions of ‘gnuplot‘. Also, a history mechanism allows previous commands to be edited and re-executed. After the command line has been edited, a newline or carriage return will enter the entire line without regard to where the cursor is positioned.

(The readline function in ‘gnuplot‘ is not the same as the readline used in GNU Bash and GNU Emacs. If the GNU version is desired, it may be selected instead of the ‘gnuplot‘ version at compile time.)

The editing commands are as follows:

```
‘Line-editing‘:

^B    moves back a single character.
^F    moves forward a single character.
^A    moves to the beginning of the line.
^E    moves to the end of the line.
^H    and DEL delete the previous character.
^D    deletes the current character.
^K    deletes from current position to the end of line.
^L,^R redraws line in case it gets trashed.
^U    deletes the entire line.
^W    deletes from the current word to the end of line.

‘History‘:

^P    moves back through history.
^N    moves forward through history.
```

On the IBM PC, the use of a TSR program such as DOSEDIT or CED may be desired for line editing. The default makefile assumes that this is the case; by default 'gnuplot' will be compiled with no line-editing capability. If you want to use 'gnuplot''s line editing, set READLINE in the makefile and add readline.obj to the link file. The following arrow keys may be used on the IBM PC and Atari versions if readline is used:

```
Left  Arrow      - same as ^B.
Right Arrow      - same as ^F.
Ctrl Left  Arrow - same as ^A.
Ctrl Right Arrow - same as ^E.
Up    Arrow      - same as ^P.
Down  Arrow      - same as ^N.
```

The Atari version of readline defines some additional key aliases:

```
Undo            - same as ^L.
Home            - same as ^A.
Ctrl Home       - same as ^E.
Esc             - same as ^U.
Help            - Section 2.6 [help], page 45 plus return.
Ctrl Help       - Section 2.6 [help], page 45.
```

1.8 Comments

Comments are supported as follows: a '#' may appear in most places in a line and 'gnuplot' will ignore the rest of the line. It will not have this effect inside quotes, inside numbers (including complex numbers), inside command substitutions, etc. In short, it works anywhere it makes sense to work.

See also 'set datafile commentschars' for specifying comment characters in data files.

1.9 Coordinates

The commands Section 2.21.2 [arrow], page 75, Section 2.21.28 [key], page 100, and Section 2.21.29 [label], page 104 allow you to draw something at an arbitrary position on the graph. This position is specified by the syntax:

```
{<system>} <x>, {<system>} <y> {,{<system>} <z>}
```

Each <system> can either be 'first', 'second', 'graph', 'screen', or 'character'.

'first' places the x, y, or z coordinate in the system defined by the left and bottom axes; 'second' places it in the system defined by the second axes (top and right); 'graph' specifies the area within the axes—0,0 is bottom left and 1,1 is top right (for splot, 0,0,0 is bottom left of plotting area; use negative z to get to the base—see Section 2.21.64 [ticslevel], page 147); 'screen' specifies the screen area (the entire area—not just the portion selected by Section 2.21.57 [size], page 129), with 0,0 at bottom left and 1,1 at top right; and 'character' gives the position in character widths and heights from the bottom left of the screen area (screen 0,0), 'character' coordinates depend on the chosen font size.

If the coordinate system for x is not specified, ‘first‘ is used. If the system for y is not specified, the one used for x is adopted.

In some cases, the given coordinate is not an absolute position but a relative value (e.g., the second position in Section 2.21.2 [arrow], page 75 ... ‘rto‘). In most cases, the given value serves as difference to the first position. If the given coordinate resides in a logarithmic axis the value is interpreted as factor. For example,

```
set logscale x
set arrow 100,5 rto 10,2
```

plots an arrow from position 100,5 to position 1000,7 since the x axis is logarithmic while the y axis is linear.

If one (or more) axis is timeseries, the appropriate coordinate should be given as a quoted time string according to the Section 2.21.67 [timefmt], page 147 format string. See Section 2.21.83 [xdata], page 151 and Section 2.21.67 [timefmt], page 147. ‘gnuplot‘ will also accept an integer expression, which will be interpreted as seconds from 1 January 2000.

1.10 Datastrings

The configuration option –enable-datastrings allows gnuplot to read and process text fields in datafiles. A text field consists of either an arbitrary string of printable characters containing no whitespace or an arbitrary string of characters, possibly including whitespace, delimited by double quotes. The following sample line from a datafile is interpreted to contain four columns, with a text field in column 3:

```
1.000 2.000 "Third column is all of this text" 4.00
```

Text fields can be positioned within a 2-D or 3-D plot using the commands:

```
plot 'datafile' using 1:2:4 with labels
splot 'datafile using 1:2:3:4 with labels
```

A column of text data can also be used to label the ticmarks along one or more of the plot axes. The example below plots a line through a series of points with (X,Y) coordinates taken from columns 3 and 4 of the input datafile. However, rather than generating regularly spaced tics along the x axis labeled numerically, gnuplot will position a tic mark along the x axis at the X coordinate of each point and label the tic mark with text taken from column 1 of the input datafile.

```
set xtics
plot 'datafile' using 3:4:xticlabels(1) with linespoints
```

There is also an option that will interpret the first entry in a column of input data as a text field, and use it as the key title for data plotted from that column. The example given below will use the first entry in column 2 to generate a title in the key box, while processing the remainder of columns 2 and 4 to draw the required line:

```
plot 'datafile' using 1:(f($2)/$4) title 2 with lines
```

See ‘set style labels‘, ‘using xticlabels‘, ‘plot title‘, Section 2.12.1.9 [using], page 61.

1.11 Environment

A number of shell environment variables are understood by 'gnuplot'. None of these are required, but may be useful.

If GNUTERM is defined, it is used as the name of the terminal type to be used. This overrides any terminal type sensed by 'gnuplot' on start-up, but is itself overridden by the .gnuplot (or equivalent) start-up file (see 'start-up') and, of course, by later explicit changes.

On Unix, AmigaOS, AtariTOS, MS-DOS and OS/2, GNUHELP may be defined to be the pathname of the HELP file (gnuplot.gih).

On VMS, the logical name GNUPLOT$HELP should be defined as the name of the help library for 'gnuplot'. The 'gnuplot' help can be put inside any system help library, allowing access to help from both within and outside 'gnuplot' if desired.

On Unix, HOME is used as the name of a directory to search for a .gnuplot file if none is found in the current directory. On AmigaOS, AtariTOS, MS-DOS, Windows and OS/2, GNUPLOT is used. On Windows, the NT-specific variable USERPROFILE is tried, too. VMS, SYS$LOGIN: is used. Type 'help start-up'.

On Unix, PAGER is used as an output filter for help messages.

On Unix, AtariTOS and AmigaOS, SHELL is used for the Section 2.22 [shell], page 163 command. On MS-DOS and OS/2, COMSPEC is used for the Section 2.22 [shell], page 163 command.

On MS-DOS, if the BGI or Watcom interface is used, PCTRM is used to tell the maximum resolution supported by your monitor by setting it to S<max. horizontal resolution>. E.g. if your monitor's maximum resolution is 800x600, then use:

```
set PCTRM=S800
```

If PCTRM is not set, standard VGA is used.

FIT_SCRIPT may be used to specify a 'gnuplot' command to be executed when a fit is interrupted—see 'fit'. FIT_LOG specifies the default filename of the logfile maintained by fit.

GNUPLOT_LIB may be used to define additional search directories for data and command files. The variable may contain a single directory name, or a list of directories separated by a platform-specific path separator, eg. ':' on Unix, or ';' on DOS/Windows/OS/2/Amiga platforms. The contents of GNUPLOT_LIB are appended to the Section 2.21.31 [loadpath], page 106 variable, but not saved with the Section 2.20 [save], page 73 and 'save set' commands.

Several gnuplot terminal drivers access TrueType fonts via the gd library. For these drivers the font search path is controlled by the environmental variable GDFONTPATH. Furthermore, a default font for these drivers may be set via the environmental variable GNUPLOT_DEFAULT_GDFONT.

The postscript terminal uses its own font search path. It is controlled by the environmental variable GNUPLOT_FONTPATH. The format is the same as for GNUPLOT_LIB. The contents of GNUPLOT_FONTPATH are appended to the Section 2.21.20 [fontpath], page 93 variable, but not saved with the Section 2.20 [save], page 73 and 'save set' commands.

1.12 Expressions

In general, any mathematical expression accepted by C, FORTRAN, Pascal, or BASIC is valid. The precedence of these operators is determined by the specifications of the C programming language. White space (spaces and tabs) is ignored inside expressions.

Complex constants are expressed as {<real>,<imag>}, where <real> and <imag> must be numerical constants. For example, {3,2} represents 3 + 2i; {0,1} represents 'i' itself. The curly braces are explicitly required here.

Note that gnuplot uses both "real" and "integer" arithmetic, like FORTRAN and C. Integers are entered as "1", "-10", etc; reals as "1.0", "-10.0", "1e1", 3.5e-1, etc. The most important difference between the two forms is in division: division of integers truncates: 5/2 = 2; division of reals does not: 5.0/2.0 = 2.5. In mixed expressions, integers are "promoted" to reals before evaluation: 5/2e0 = 2.5. The result of division of a negative integer by a positive one may vary among compilers. Try a test like "print -5/2" to determine if your system chooses -2 or -3 as the answer.

The integer expression "1/0" may be used to generate an "undefined" flag, which causes a point to ignored; the 'ternary' operator gives an example.

The real and imaginary parts of complex expressions are always real, whatever the form in which they are entered: in {3,2} the "3" and "2" are reals, not integers.

Gnuplot can also perform simple operations on strings and string variables. For example, the expression ("A" . "B" eq "AB") evaluates as true, illustrating the string concatenation operator and the string equality operator.

A string which contains a numerical value is promoted to the corresponding integer or real value if used in a numerical expression. Thus ("3" + "4" == 7) and (6.78 == "6.78") both evaluate to true. An integer, but not a real or complex value, is promoted to a string if used in string concatenation. A typical case is the use of integers to construct file names or other strings; e.g. ("file" . 4 eq "file4") is true.

Substrings can be specified using a postfixed range descriptor [beg:end]. For example, "ABCDEF"[3:4] == "CD" and "ABCDEF"[4:*] == "DEF" The syntax "string"[beg:end] is exactly equivalent to calling the built-in string-valued function substr("string",beg,end), except that you cannot omit either beg or end from the function call.

1.12.1 Functions

The functions in 'gnuplot' are the same as the corresponding functions in the Unix math library, except that all functions accept integer, real, and complex arguments, unless otherwise noted.

For those functions that accept or return angles that may be given in either degrees or radians (sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), atan2(x) and arg(z)), the unit may be selected by Section 2.21.1 [angles], page 74, which defaults to radians.

1.12.1.1 abs

The 'abs(x)' function returns the absolute value of its argument. The returned value is of the same type as the argument.

For complex arguments, abs(x) is defined as the length of x in the complex plane [i.e., sqrt(real(x)**2 + imag(x)**2)].

1.12.1.2 acos

The ‘acos(x)‘ function returns the arc cosine (inverse cosine) of its argument. ‘acos‘ returns its argument in radians or degrees, as selected by Section 2.21.1 [angles], page 74.

1.12.1.3 acosh

The ‘acosh(x)‘ function returns the inverse hyperbolic cosine of its argument in radians.

1.12.1.4 arg

The ‘arg(x)‘ function returns the phase of a complex number in radians or degrees, as selected by Section 2.21.1 [angles], page 74.

1.12.1.5 asin

The ‘asin(x)‘ function returns the arc sin (inverse sin) of its argument. ‘asin‘ returns its argument in radians or degrees, as selected by Section 2.21.1 [angles], page 74.

1.12.1.6 asinh

The ‘asinh(x)‘ function returns the inverse hyperbolic sin of its argument in radians.

1.12.1.7 atan

The ‘atan(x)‘ function returns the arc tangent (inverse tangent) of its argument. ‘atan‘ returns its argument in radians or degrees, as selected by Section 2.21.1 [angles], page 74.

1.12.1.8 atan2

The ‘atan2(y,x)‘ function returns the arc tangent (inverse tangent) of the ratio of the real parts of its arguments. Section 1.12.1.8 [atan2], page 14 returns its argument in radians or degrees, as selected by Section 2.21.1 [angles], page 74, in the correct quadrant.

1.12.1.9 atanh

The ‘atanh(x)‘ function returns the inverse hyperbolic tangent of its argument in radians.

1.12.1.10 besj0

The ‘besj0(x)‘ function returns the j0th Bessel function of its argument. Section 1.12.1.10 [besj0], page 14 expects its argument to be in radians.

1.12.1.11 besj1

The ‘besj1(x)‘ function returns the j1st Bessel function of its argument. Section 1.12.1.11 [besj1], page 14 expects its argument to be in radians.

1.12.1.12 besy0

The ‘besy0(x)‘ function returns the y0th Bessel function of its argument. Section 1.12.1.12 [besy0], page 14 expects its argument to be in radians.

1.12.1.13 besy1

The ‘besy1(x)‘ function returns the y1st Bessel function of its argument. Section 1.12.1.13 [besy1], page 14 expects its argument to be in radians.

1.12.1.14 ceil

The ‘ceil(x)‘ function returns the smallest integer that is not less than its argument. For complex numbers, Section 1.12.1.14 [ceil], page 15 returns the smallest integer not less than the real part of its argument.

1.12.1.15 cos

The ‘cos(x)‘ function returns the cosine of its argument. ‘cos‘ accepts its argument in radians or degrees, as selected by Section 2.21.1 [angles], page 74.

1.12.1.16 cosh

The ‘cosh(x)‘ function returns the hyperbolic cosine of its argument. Section 1.12.1.16 [cosh], page 15 expects its argument to be in radians.

1.12.1.17 erf

The ‘erf(x)‘ function returns the error function of the real part of its argument. If the argument is a complex value, the imaginary component is ignored. See Section 1.12.1.18 [erfc], page 15, Section 1.12.1.23 [inverf], page 16, and Section 1.12.1.32 [norm], page 16.

1.12.1.18 erfc

The ‘erfc(x)‘ function returns 1.0 - the error function of the real part of its argument. If the argument is a complex value, the imaginary component is ignored. See ‘erf‘, Section 1.12.1.23 [inverf], page 16, and Section 1.12.1.32 [norm], page 16.

1.12.1.19 exp

The ‘exp(x)‘ function returns the exponential function of its argument (‘e‘ raised to the power of its argument). On some implementations (notably suns), exp(-x) returns undefined for very large x. A user-defined function like safe(x) = x<-100 ? 0 : exp(x) might prove useful in these cases.

1.12.1.20 floor

The ‘floor(x)‘ function returns the largest integer not greater than its argument. For complex numbers, Section 1.12.1.20 [floor], page 15 returns the largest integer not greater than the real part of its argument.

1.12.1.21 gamma

The ‘gamma(x)‘ function returns the gamma function of the real part of its argument. For integer n, gamma(n+1) = n!. If the argument is a complex value, the imaginary component is ignored.

1.12.1.22 ibeta

The ‘ibeta(p,q,x)‘ function returns the incomplete beta function of the real parts of its arguments. p, q > 0 and x in [0:1]. If the arguments are complex, the imaginary components are ignored.

1.12.1.23 inverf

The ‘inverf(x)‘ function returns the inverse error function of the real part of its argument. See ‘erf‘ and Section 1.12.1.26 [invnorm], page 16.

1.12.1.24 igamma

The ‘igamma(a,x)‘ function returns the incomplete gamma function of the real parts of its arguments. a > 0 and x >= 0. If the arguments are complex, the imaginary components are ignored.

1.12.1.25 imag

The ‘imag(x)‘ function returns the imaginary part of its argument as a real number.

1.12.1.26 invnorm

The ‘invnorm(x)‘ function returns the inverse normal distribution function of the real part of its argument. See Section 1.12.1.32 [norm], page 16.

1.12.1.27 int

The ‘int(x)‘ function returns the integer part of its argument, truncated toward zero.

1.12.1.28 lambertw

The lambertw function returns the value of the principal branch of Lambert’s W function, which is defined by the equation (W(z)*exp(W(z))=z. z must be a real number with z >= -exp(-1).

1.12.1.29 lgamma

The ‘lgamma(x)‘ function returns the natural logarithm of the gamma function of the real part of its argument. If the argument is a complex value, the imaginary component is ignored.

1.12.1.30 log

The ‘log(x)‘ function returns the natural logarithm (base ‘e‘) of its argument. See Section 1.12.1.31 [log10], page 16.

1.12.1.31 log10

The ‘log10(x)‘ function returns the logarithm (base 10) of its argument.

1.12.1.32 norm

The ‘norm(x)‘ function returns the normal distribution function (or Gaussian) of the real part of its argument. See Section 1.12.1.26 [invnorm], page 16, ‘erf‘ and Section 1.12.1.18 [erfc], page 15.

1.12.1.33 rand

‘rand(0)‘ returns a pseudo random number in the interval [0:1] generated

```
from the current value of two internal 32-bit seeds.
```

‘rand(-1)‘ resets both seeds to a standard value. ‘rand(x)‘ for x>0 sets both seeds to a value based on the value of x. ‘rand({x,y})‘ for x>0 sets seed1 to x and seed2 to y. Note: This behavior has changed starting with gnuplot version 3.8l. Older scripts that expected rand(x>0) to produce sequential pseudo-random numbers from the same seeded sequence must be changed to call rand(0) instead.

1.12.1.34 real

The ‘real(x)‘ function returns the real part of its argument.

1.12.1.35 sgn

The ‘sgn(x)‘ function returns 1 if its argument is positive, -1 if its argument is negative, and 0 if its argument is 0. If the argument is a complex value, the imaginary component is ignored.

1.12.1.36 sin

The ‘sin(x)‘ function returns the sine of its argument. ‘sin‘ expects its argument to be in radians or degrees, as selected by Section 2.21.1 [angles], page 74.

1.12.1.37 sinh

The ‘sinh(x)‘ function returns the hyperbolic sine of its argument. Section 1.12.1.37 [sinh], page 17 expects its argument to be in radians.

1.12.1.38 sqrt

The ‘sqrt(x)‘ function returns the square root of its argument.

1.12.1.39 tan

The ‘tan(x)‘ function returns the tangent of its argument. ‘tan‘ expects its argument to be in radians or degrees, as selected by Section 2.21.1 [angles], page 74.

1.12.1.40 tanh

The ‘tanh(x)‘ function returns the hyperbolic tangent of its argument. Section 1.12.1.40 [tanh], page 17 expects its argument to be in radians.

1.12.1.41 gprintf

‘gprintf("format",x)‘ applies gnuplot’s own format specifiers to the single variable x and returns the resulting string. If you want standard C-language format specifiers, you must instead use ‘sprintf("format",x)‘. See ‘format specifiers‘.

1.12.1.42 sprintf

‘sprintf("format",var1,var2,...)‘ applies standard C-language format specifiers to multiple arguments (10 max) and returns the resulting string. If you want to use gnuplot’s own format specifiers, you must instead call ‘gprintf()‘. For information on sprintf format specifiers, please see standard C-language documentation or the unix sprintf man page.

1.12.1.43 strlen

‘strlen("string")‘ returns the number of characters in the string.

1.12.1.44 strstrt

'strstrt("string","key")' searches for the character string "key" in "string" and returns the index to the first character of "key". If "key" is not found, returns 0. Similar to C library function strstr except that it returns an index rather than a string pointer. strstrt("hayneedlestack","needle") = 4.

1.12.1.45 substr

'substr("string",beg,end)' returns the substring consisting of characters beg through end of the original string. This is exactly equivalent to the expression "string"[beg:end] except that you do not have the option of ommitting beg or end.

1.12.1.46 system

'system("command")' executes "command" using the standard shell and returns the resulting character stream from stdout as string variable. One optional trailing newline is ignored.

This can be used to import external functions into gnuplot scripts using 'f(x) = real(system(sprintf("somecommand %f", x)))'.

1.12.1.47 word

'word("string",n)' returns the nth word in string. For example, 'word("one two three",2)' returns the string "two".

1.12.1.48 words

'words("string")' returns the number of words in string. For example, 'words(" a b c d")' returns the 4.

1.12.1.49 column

'column(x)' may be used only in expressions as part of Section 2.12.1.9 [using], page 61 manipulations to fits or datafile plots. It evaluates to the numerical value of the contents of column x. See Section 2.12.1.9 [using], page 61.

1.12.1.50 defined

'defined(X)' returns 1 if a variable named X has been defined, otherwise it returns 0.

1.12.1.51 stringcolumn

'stringcolumn(x)' may be used only in expressions as part of Section 2.12.1.9 [using], page 61 manipulations to fits or datafile plots. It returns the content of column x as a string variable. See Section 2.12.1.9 [using], page 61.

1.12.1.52 timecolumn

'timecolumn(x)' may be used only in expressions as part of Section 2.12.1.9 [using], page 61 manipulations to fits or datafile plots. See Section 2.12.1.9 [using], page 61.

It reads the data starting at that column as a time/date value and returns its value in gnuplot's internal time representation of "seconds since the millennium".

To find the right Section 2.21.67 [timefmt], page 147 string to use, Section 1.12.1.52 [timecolumn], page 18 searches for a Section 2.12.1.9 [using], page 61 specification with the same column number as its argument. If one is found, Section 2.21.67 [timefmt], page 147 pattern of the target axis for this specifier is used. Otherwise, Section 1.12.1.52 [timecolumn], page 18 chooses the x axis Section 2.21.67 [timefmt], page 147 per default.

1.12.1.53 tm_hour

The Section 1.12.1.53 [tm_hour], page 19 function interprets its argument as a time, in seconds from 1 Jan 2000. It returns the hour (an integer in the range 0–23) as a real.

1.12.1.54 tm_mday

The Section 1.12.1.54 [tm_mday], page 19 function interprets its argument as a time, in seconds from 1 Jan 2000. It returns the day of the month (an integer in the range 1–31) as a real.

1.12.1.55 tm_min

The Section 1.12.1.55 [tm_min], page 19 function interprets its argument as a time, in seconds from 1 Jan 2000. It returns the minute (an integer in the range 0–59) as a real.

1.12.1.56 tm_mon

The Section 1.12.1.56 [tm_mon], page 19 function interprets its argument as a time, in seconds from 1 Jan 2000. It returns the month (an integer in the range 0–11) as a real.

1.12.1.57 tm_sec

The Section 1.12.1.57 [tm_sec], page 19 function interprets its argument as a time, in seconds from 1 Jan 2000. It returns the second (an integer in the range 0–59) as a real.

1.12.1.58 tm_wday

The Section 1.12.1.58 [tm_wday], page 19 function interprets its argument as a time, in seconds from 1 Jan 2000. It returns the day of the week (an integer in the range 0–6) as a real.

1.12.1.59 tm_yday

The Section 1.12.1.59 [tm_yday], page 19 function interprets its argument as a time, in seconds from 1 Jan 2000. It returns the day of the year (an integer in the range 1–366) as a real.

1.12.1.60 tm_year

The Section 1.12.1.60 [tm_year], page 19 function interprets its argument as a time, in seconds from 1 Jan 2000. It returns the year (an integer) as a real.

1.12.1.61 valid

'valid(x)' may be used only in expressions as part of Section 2.12.1.9 [using], page 61 manipulations to fits or datafile plots. See Section 2.12.1.9 [using], page 61.

See also airfoil.dem: use of functions and complex variables for airfoils demo.

1.12.1.62 Random number generator

The behavior of the built-in function 'rand(x)' has changed as of version 3.8l. Older scripts that expected rand(x>0) to produce sequential pseudo-random numbers from the same seeded sequence must be changed to call rand(0) instead. The current behavior is as follows:

```
 'rand(0)'  returns a pseudo random number in the inter-
val [0:1] generated
            from the current value of two internal 32-bit seeds.
 'rand(-1)' resets both seeds to a standard value.
 'rand(x)'  for x>0 sets both seeds to a value based on the value of x.
 'rand({x,y})' for x>0 sets seed1 to x and seed2 to y.
```

1.12.2 Operators

The operators in 'gnuplot' are the same as the corresponding operators in the C programming language, except that all operators accept integer, real, and complex arguments, unless otherwise noted. The ** operator (exponentiation) is supported, as in FORTRAN.

Parentheses may be used to change order of evaluation.

1.12.2.1 Unary

The following is a list of all the unary operators and their usages:

```
      Symbol      Example    Explanation
        -           -a          unary minus
        +           +a          unary plus (no-operation)
        ~           ~a        * one's complement
        !           !a        * logical negation
        !           a!        * factorial
        $           $3        * call arg/column during Section 2.12.1.9
    [using], page 61 manipulation
```

(*) Starred explanations indicate that the operator requires an integer argument.

Operator precedence is the same as in Fortran and C. As in those languages, parentheses may be used to change the order of operation. Thus -2**2 = -4, but (-2)**2 = 4.

The factorial operator returns a real number to allow a greater range.

1.12.2.2 Binary

The following is a list of all the binary operators and their usages:

```
      Symbol       Example      Explanation
        **          a**b          exponentiation
        *           a*b           multiplication
        /           a/b           division
        %           a%b         * modulo
        +           a+b           addition
        -           a-b           subtraction
        ==          a==b          equality
        !=          a!=b          inequality
```

```
<          a<b         less than
<=         a<=b        less than or equal to
>          a>b         greater than
>=         a>=b        greater than or equal to
&          a&b       * bitwise AND
^          a^b       * bitwise exclusive OR
|          a|b       * bitwise inclusive OR
&&         a&&b      * logical AND
||         a||b      * logical OR
.          A.B         string concatenation
eq         A eq B      string equality
ne         A ne B      string inequality
```

(*) Starred explanations indicate that the operator requires integer arguments. Capital letters A and B indicate that the operator requires string arguments.

Logical AND (&&) and OR (||) short-circuit the way they do in C. That is, the second ‘&&‘ operand is not evaluated if the first is false; the second ‘||‘ operand is not evaluated if the first is true.

1.12.2.3 Ternary

There is a single ternary operator:

```
Symbol       Example      Explanation
  ?:          a?b:c     ternary operation
```

The ternary operator behaves as it does in C. The first argument (a), which must be an integer, is evaluated. If it is true (non-zero), the second argument (b) is evaluated and returned; otherwise the third argument (c) is evaluated and returned.

The ternary operator is very useful both in constructing piecewise functions and in plotting points only when certain conditions are met.

Examples:

Plot a function that is to equal sin(x) for 0 <= x < 1, 1/x for 1 <= x < 2, and undefined elsewhere:

```
f(x) = 0<=x && x<1 ? sin(x) : 1<=x && x<2 ? 1/x : 1/0
plot f(x)
```

Note that ‘gnuplot‘ quietly ignores undefined values, so the final branch of the function (1/0) will produce no plottable points. Note also that f(x) will be plotted as a continuous function across the discontinuity if a line style is used. To plot it discontinuously, create separate functions for the two pieces. (Parametric functions are also useful for this purpose.)

For data in a file, plot the average of the data in columns 2 and 3 against the datum in column 1, but only if the datum in column 4 is non-negative:

```
plot ’file’ using 1:( $4<0 ? 1/0 : ($2+$3)/2 )
```

Please see Section 2.12.1.9 [using], page 61 for an explanation of the Section 2.12.1.9 [using], page 61 syntax.

1.12.3 Gnuplot-defined variables

The variable ‘pi‘ is defined to be pi, see

```
print pi
```

Additionally, gnuplot may define some variables under various operations.

Working with interactive terminals with ‘mouse‘ functionality defines variables with names that begin "MOUSE_", see Section 2.21.72 [variables], page 149 for details.

The ‘fit‘ mechanism uses several variables with names that begin "FIT_". It is safest to avoid using such names. "FIT_LIMIT", however, is one that you may wish to redefine. Under ‘set fit errorvariables‘, the error for each fitted parameter will be stored in a variable named like the parameter, but with "_err" appended. See the documentation on ‘fit‘ for details.

See Section 2.21.72 [variables], page 149, Section 2.21.72 [variables], page 149, and ‘fit‘.

1.12.4 User-defined variables and functions

New user-defined variables and functions of one through five variables may be declared and used anywhere, including on the ‘plot‘ command itself.

User-defined function syntax:

```
      <func-
name>( <dummy1> {,<dummy2>} ... {,<dummy5>} ) = <expression>
```

where <expression> is defined in terms of <dummy1> through <dummy5>.

User-defined variable syntax:

```
<variable-name> = <constant-expression>
```

Examples:

```
w = 2
q = floor(tan(pi/2 - 0.1))
f(x) = sin(w*x)
sinc(x) = sin(pi*x)/(pi*x)
delta(t) = (t == 0)
ramp(t) = (t > 0) ? t : 0
min(a,b) = (a < b) ? a : b
comb(n,k) = n!/(k!*(n-k)!)
len3d(x,y,z) = sqrt(x*x+y*y+z*z)
plot f(x) = sin(x*a), a = 0.2, f(x), a = 0.4, f(x)

file = "mydata.inp"
file(n) = sprintf("run_%d.dat",n)
```

The final two examples illustrate a user-defined string variable and a user-defined string function.

Note that the variable ‘pi‘ is already defined. But it is in no way magic; you may redefine it to be whatever you like. Some other variables may be defined under various gnuplot operations like mousing in interactive terminals or fitting; see Section 2.21.72 [variables], page 149 for details.

You can check for existence of a given variable by the defined(v) expression, for example

```
a = 10
if (defined(a)) print "a is defined"
if (!defined(b)) print "b is not defined"
```

Valid names are the same as in most programming languages: they must begin with a letter, but subsequent characters may be letters, digits, "$", or "_".

See ‘show functions‘, ‘functions‘, Section 2.21.72 [variables], page 149.

1.13 Glossary

Throughout this document an attempt has been made to maintain consistency of nomenclature. This cannot be wholly successful because as ‘gnuplot‘ has evolved over time, certain command and keyword names have been adopted that preclude such perfection. This section contains explanations of the way some of these terms are used.

A "page" or "screen" is the entire area addressable by ‘gnuplot‘. On a monitor, it is the full screen; on a plotter, it is a single sheet of paper.

A screen may contain one or more "plots". A plot is defined by an abscissa and an ordinate, although these need not actually appear on it, as well as the margins and any text written therein.

A plot contains one "graph". A graph is defined by an abscissa and an ordinate, although these need not actually appear on it.

A graph may contain one or more "lines". A line is a single function or data set. "Line" is also a plotting style. The word will also be used in sense "a line of text". Presumably the context will remove any ambiguity.

The lines on a graph may have individual names. These may be listed together with a sample of the plotting style used to represent them in the "key", sometimes also called the "legend".

The word "title" occurs with multiple meanings in ‘gnuplot‘. In this document, it will always be preceded by the adjective "plot", "line", or "key" to differentiate among them.

A 2-d graph may have up to four labelled axes. The names of the four axes for these usages are "x" for the axis along the bottom border of the plot, "y" for the left border, "x2" for the top border, and "y2" for the right border.

A 3-d graph may have up to three labelled axes – "x", "y" and "z". It is not possible to say where on the graph any particular axis will fall because you can change the direction from which the graph is seen with Section 2.21.74 [view], page 150.

When discussing data files, the term "record" will be resurrected and used to denote a single line of text in the file, that is, the characters between newline or end-of-record characters. A "point" is the datum extracted from a single record. A "datablock" is a set of points from consecutive records, delimited by blank records. A line, when referred to in the context of a data file, is a subset of a datablock.

1.14 linetype, colors, and styles

Each gnuplot terminal type provides a set of distinct "linetypes". These may differ in color, in thickness, in dot/dash pattern, or in some combination of color and dot/dash. The default linetypes for a particular terminal can be previewed by issuing the Section 2.25 [test], page 168 command after setting the terminal type. The pre-defined colors and dot/dash patterns are not guaranteed to be consistent for all terminal types, but all terminals use the special linetype -1 to mean a solid line in the primary foreground color (normally black). By default, successive functions or datafiles plotted by a single command will be assigned successive linetypes. You can override this default by specifying a particular linetype for any function, datafile, or plot element.

Examples:

```
      plot "foo", "bar"                 # plot two files using line-
types 1, 2
      plot sin(x) linetype 4            # terminal-specific line-
type color 4
      plot sin(x) lt -1                 # black
```

For many terminal types it is also possible to assign user-defined colors using explicit rgb (red, green, blue) values, named colors, or color values that refer to the current PM3D palette.

Examples:

```
      plot sin(x) lt rgb "violet"       # one of gnuplot's named colors
      plot sin(x) lt rgb "#FF00FF"      # ex-
plicit RGB triple in hexadecimal
      plot sin(x) lt palette cb -45     # whatever color corre-
sponds to -45
                                        # in the cur-
rent cbrange of the palette
      plot sin(x) lt palette frac 0.3   # frac-
tional value along the palette
```

See Section 2.21.50.7 [colornames], page 126, Section 2.21.50 [palette], page 120, Section 2.21.118 [cbrange], page 162.

For terminals that support dot/dash patterns, each default linetype has both a dot-dash pattern and a default color. However, you can override the default color by using the keyword ‘linecolor‘, abbreviated ‘lc‘. For example, the postscript terminal provides a dashed blue line as linetype 3. The plot commands below use this same dash pattern for three plots, one in blue (the default), another in red (the default for linetype 1), and a third in gold.

Example:

```
      set term postscript dashed color
      plot 'foo' lt 3, 'baz' lt 3 linecolor 1, 'bar' lt 3 lc rgb 'gold'
```

Lines can have additional properties such as linewidth. You can associate these various properties, as well as equivalent properties for point symbols, into user-defined "line styles" using the command 'set style line'. Once you have defined a linestyle, you can use it in a plot command to control the appearance of one or more plot elements.

Examples:

```
      # define a new line style with terminal-independent color cyan,
      # linewidth 3, and associated point type 6 (a cir-
  cle with a dot in it).
      set style line 5 lt rgb "cyan" lw 3 pt 6
      plot sin(x) with linespoints ls 5          # user-
  defined line style 5
```

See 'linestyle', 'set style line'.

1.14.1 colorspec

Many commands allow you to specify a linetype with an explicit color. This option is only possible for terminals that support RGB color or pm3d palettes.

Syntax:

```
      ... {linetype | lt} <colorspec>
```

where <colorspec> has one of the following forms:

```
      rgbcolor "colorname"
      rgbcolor "#RRGGBB"
      rgbcolor variable
      palette frac <val>      # <val> runs from 0 to 1
      palette cb <value>      # <val> lies within cbrange
      palette z
```

"colorname" refers to one of the color names built in to gnuplot. For a list of the available names, see Section 2.21.50.7 [colornames], page 126.

"#RRGGBB" is a hexadecimal constant preceded by the "#" symbol. The RRGGBB represents the red, green, and blue components of the color, each on a scale from 0 - 255. For example, magenta = full-scale red + full-scale blue would be represented by #FF00FF, which is the hexadecimal representation of (255 << 16) + (0 << 8) + (255).

"rgb variable" requires an additional column in the Section 2.12.1.9 [using], page 61 specifier, and is only available in 3D plotting mode (splot). The extra column is interpreted as a 24-bit packed RGB triple. These are most easily specified in a data file as hexidecimal values (see above).

Example:

```
      rgb(r,g,b) = 65536 * int(r) + 256 * int(g) + int(b)
      splot "data" us-
  ing 1:2:3:(rgb($1,$2,$3)) with points lc rgb variable
```

The color palette is a linear gradient of colors that smoothly maps a single numerical value onto a particular color. Two such mappings are always in effect. 'palette frac' maps a fractional value between 0 and 1 onto the full range of the color palette. 'palette cb' maps the range of the color axis onto the same palette. See Section 2.21.118 [cbrange], page 162. See also 'set colorbox'. You can use either of these to select a constant color from the current palette.

"palette z" maps the z value of each plot segment or plot element into the cbrange mapping of the palette. This allows smoothly-varying color along a 3d line or surface. This option applies only to 3D plots (splot).

1.15 mouse input

The 'x11', 'pm', 'windows', and 'ggi' terminals allow interaction with the current plot using the mouse. They also support the definition of hotkeys to activate pre-defined functions by hitting a single key while the mouse focus is in the active plot window. It is even possible to combine mouse input with 'batch' command scripts, by invoking the command 'pause mouse' and then using the mouse variables returned by mouse clicking as parameters for subsequent scripted actions. See Section 1.15.1 [bind], page 26 and Section 2.21.72 [variables], page 149. See also the command 'set mouse'.

1.15.1 bind

The Section 1.15.1 [bind], page 26 allows defining or redefining a hotkey, i.e. a sequence of gnuplot commands which will be executed when a certain key or key sequence is pressed while the driver's window has the input focus. Note that Section 1.15.1 [bind], page 26 is only available if gnuplot was compiled with 'mouse' support and it is used by all mouse-capable terminals. Bindings overwrite the builtin bindings (like in every real editor), except <space> and 'q' which cannot be rebound. Mouse buttons cannot be rebound.

Note that multikey-bindings with modifiers have to be quoted.

Normally hotkeys are only recognized when the currently active plot window has focus. 'bind allwindows <key> ...' (short form: 'bind all <key> ...') causes the binding for <key> to apply to all gnuplot plot windows, active or not. In this case gnuplot variable MOUSE_KEY_WINDOW is set to the ID of the originating window, and may be used by the bound command.

Syntax:

```
bind {allwindows} [<key-sequence>] ["<gnuplot commands>"]
bind!
```

Examples:

- set bindings:

```
bind a "replot"
bind "ctrl-a" "plot x*x"
bind "ctrl-alt-a" 'print "great"'
bind Home "set view 60,30; replot"
bind all Home 'print "This is window ",MOUSE_KEY_WINDOW'
```

- show bindings:

```
bind "ctrl-a"          # shows the binding for ctrl-a
bind                   # shows all bindings
```

- remove bindings:

```
bind "ctrl-alt-a" ""   # removes binding for ctrl-alt-a
                         (note that builtins cannot be removed)
bind!                  # installs default (builtin) bindings
```

- bind a key to toggle something:

```
  v=0
  bind "ctrl-r" "v=v+1;if(v%2)set term x11 no-
raise; else set term x11 raise"
```

Modifiers (ctrl / alt) are case insensitive, keys not:

```
ctrl-alt-a == CtRl-alT-a
ctrl-alt-a != ctrl-alt-A
```

List of modifiers (alt == meta):

```
ctrl, alt
```

List of supported special keys:

```
   "BackSpace", "Tab", "Linefeed", "Clear", "Re-
turn", "Pause", "Scroll_Lock",
   "Sys_Req", "Es-
cape", "Delete", "Home", "Left", "Up", "Right", "Down",
   "PageUp", "PageDown", "End", "Begin",

   "KP_Space", "KP_Tab", "KP_Enter", "KP_F1", "KP_F2", "KP_F3", "KP_F4",
   "KP_Home", "KP_Left", "KP_Up", "KP_Right", "KP_Down", "KP_PageUp",
   "KP_PageDown", "KP_End", "KP_Begin", "KP_Insert", "KP_Delete", "KP_Equal",
   "KP_Multiply", "KP_Add", "KP_Separator", "KP_Subtract", "KP_Decimal",
   "KP_Divide",

   "KP_1" - "KP_9", "F1" - "F12"
```

See also help for 'mouse' and Section 2.8 [if], page 46.

1.15.2 Mouse variables

When mousing is active, clicking in the active window will set several user variables that can be accessed from the gnuplot command line. The coordinates of the mouse at the time of the click are stored in MOUSE_X MOUSE_Y MOUSE_X2 and MOUSE_Y2. The mouse button clicked, and any meta-keys active at that time, are stored in MOUSE_BUTTON MOUSE_SHIFT MOUSE_ALT and MOUSE_CTRL. These variables are set to undefined at the start of every plot, and only become defined in the event of a mouse click in the active plot window. To determine from a script if the mouse has been clicked in the active plot window, it is sufficient to test for any one of these variables being defined.

```
plot 'something'
pause mouse
if (defined(MOUSE_BUTTON)) call 'something_else'; \
else print "No mouse click."
```

It is also possible to track keystrokes in the plot window using the mousing code.

```
plot 'something'
pause mouse keypress
print "Keystroke ", MOUSE_KEY, " at ", MOUSE_X, " ", MOUSE_Y
```

When 'pause mouse keypress' is terminated by a keypress, then MOUSE_KEY will contain the ascii character value of the key that was pressed. MOUSE_CHAR will contain the character itself as a string variable. If the pause command is terminated abnormally (e.g. by ctrl-C or by externally closing the plot window) then MOUSE_KEY will equal -1.

1.16 Plotting

There are three 'gnuplot' commands which actually create a plot: 'plot', 'splot' and Section 2.17 [replot], page 71. 'plot' generates 2-d plots, 'splot' generates 3-d plots (actually 2-d projections, of course), and Section 2.17 [replot], page 71 appends its arguments to the previous 'plot' or 'splot' and executes the modified command.

Much of the general information about plotting can be found in the discussion of 'plot'; information specific to 3-d can be found in the 'splot' section.

'plot' operates in either rectangular or polar coordinates – see 'set polar' for details of the latter. 'splot' operates only in rectangular coordinates, but the Section 2.21.35 [mapping], page 108 command allows for a few other coordinate systems to be treated. In addition, the Section 2.12.1.9 [using], page 61 option allows both 'plot' and 'splot' to treat almost any coordinate system you'd care to define.

'plot' also lets you use each of the four borders – x (bottom), x2 (top), y (left) and y2 (right) – as an independent axis. The 'axes' option lets you choose which pair of axes a given function or data set is plotted against. A full complement of 'set' commands exists to give you complete control over the scales and labelling of each axis. Some commands have the name of an axis built into their names, such as Section 2.21.85 [xlabel], page 152. Other commands have one or more axis names as options, such as 'set logscale xy'. Commands and options controlling the z axis have no effect on 2-d graphs.

'splot' can plot surfaces and contours in addition to points and/or lines. In addition to 'splot', see Section 2.21.27 [isosamples], page 100 for information about defining the grid for a 3-d function; Section 2.21.14 [datafile], page 86 for information about the requisite file structure for 3-d data values; and Section 2.21.12 [contour], page 86 and Section 2.21.10 [cntrparam], page 83 for information about contours.

In 'splot', control over the scales and labels of the axes are the same as with 'plot', except that commands and options controlling the x2 and y2 axes have no effect whereas of course those controlling the z axis do take effect.

1.17 Start-up

When ‘gnuplot‘ is run, it looks for an initialization file to load. This file is called ‘.gnuplot‘ on Unix and AmigaOS systems, and ‘GNUPLOT.INI‘ on other systems. If this file is not found in the current directory, the program will look for it in the HOME directory (under AmigaOS, Atari(single)TOS, MS-DOS, Windows and OS/2, the environment variable ‘GNUPLOT‘ should contain the name of this directory; on Windows NT, it will use ‘USERPROFILE‘ if GNUPLOT isn’t defined). Note: if NOCWDRC is defined during the installation, ‘gnuplot‘ will not read from the current directory.

If the initialization file is found, ‘gnuplot‘ executes the commands in it. These may be any legal ‘gnuplot‘ commands, but typically they are limited to setting the terminal and defining frequently-used functions or variables.

1.18 String constants and string variables

A new feature in gnuplot 4.1 is the introduction of string variables and string functions. Most gnuplot commands that previously required a string constant will now also accept a string variable, a string expression, or a function that returns a string. For example, the following four methods of creating a plot all result in the same plot title:

```
four = "4"
graph4 = "Title for plot #4"
graph(n) = sprintf("Title for plot #%d",n)

plot 'data.4' title "Title for plot #4"
plot 'data.4' title graph4
plot 'data.4' title "Title for plot #".four
plot 'data.4' title graph(4)
```

Since integers are promoted to strings when operated on by the string concatenation operator, the following method also works:

```
N = 4
plot 'data.'.N title "Title for plot #".N
```

In general, elements on the command line will only be evaluated as possible string variables if they are not otherwise recognizable as part of the normal gnuplot syntax. So the following sequence of commands is legal, although probably should be avoided so as not to cause confusion:

```
plot = "my_datafile.dat"
title = "My Title"
plot plot title title
```

There are three binary operators that require string operands: the string concatenation operator ".", the string equality operator "eq" and the string inequality operator "ne". The following example will print TRUE.

```
if ("A"."B" eq "AB") print "TRUE"
```

See also the two string formatting functions ‘gprintf‘ and Section 1.12.1.42 [sprintf], page 17.

Substrings can be specified by appending a range specifier to any string, string variable, or string-valued function. The range specifier has the form [begin:end], where begin is the index of the first character of the substring and end is the index of the last character of the substring. The first character has index 1. The begin or end fields may be empty, or contain ’*’, to indicate the true start or end of the original string. E.g. str[:] and str[*:*] both describe the full string str.

1.19 Substitution and Command line macros

When a command line to gnuplot is first read, i.e. before it is interpreted or executed, two forms of lexical substitution are performed. These are triggered by the presence of text in backquotes (ascii character 96) or preceded by `@` (ascii character 64).

1.19.1 Substitution of system commands in backquotes

Command-line substitution is specified by a system command enclosed in backquotes. This command is spawned and the output it produces replaces the backquoted text on the command line. Some implementations also support pipes; see Section 2.12.1.7 [special-filenames], page 59.

Command-line substitution can be used anywhere on the ‘gnuplot‘ command line, except inside strings delimited by single quotes.

Example:

This will run the program ‘leastsq‘ and replace ‘leastsq‘ (including backquotes) on the command line with its output:

```
f(x) = `leastsq`
```

or, in VMS

```
f(x) = `run leastsq`
```

These will generate labels with the current time and userid:

```
set label "generated on `date +%Y-%m-%d` by `whoami`" at 1,1
set timestamp "generated on %Y-%m-%d by `whoami`"
```

1.19.2 Substitution of string variables as macros

Substitution of command line macros is disabled by default, but may be enabled using the Section 2.21.34 [macros], page 108 command. If macro substitution is enabled, the character `@` is used to trigger substitution of the current value of a string variable into the command line. The text in the string variable may contain any number of lexical elements. This allows string variables to be used as command line macros. Only string constants may be expanded using this mechanism, not string-valued expressions. For example:

```
set macros
style1 = "lines lt 4 lw 2"
style2 = "points lt 3 pt 5 ps 2"
```

```
range1 = "using 1:3"
range2 = "using 1:5"
plot "foo" @range1 with @style1, "bar" @range2 with @style2
```

The line containing @ symbols is expanded on input, so that by the time it is executed the effect is identical to having typed in full

```
plot "foo" using 1:3 with lines lt 4 lw 2, \
     "bar" using 1:5 with points lt 3 pt 5 ps 2
```

Macro expansion does not occur inside either single or double quotes. However macro expansion does occur inside backquotes.

1.19.3 String variables, macros, and command line substitution

The interaction of string variables, backquotes and macro substitution is somewhat complicated. Backquotes do not block macro substitution, so

```
filename = "mydata.inp"
lines = ` wc --lines @filename | sed "s/ .*//" `
```

results in the number of lines in mydata.inp being stored in the integer variable lines. And double quotes do not block backquote substitution, so

```
mycomputer = "`uname -n`"
```

results in the string returned by the system command `uname -n` being stored in the string variable mycomputer.

However, macro substitution is not performed inside double quotes, so you cannot define a system command as a macro and then use both macro and backquote substitution at the same time.

```
machine_id = "uname -n"
mycomputer = "`@machine_id`"  # doesn't work!!
```

This fails because the double quotes prevent @machine_id from being interpreted as a macro. To store a system command as a macro and execute it later you must instead include the backquotes as part of the macro itself. This is accomplished by defining the macro as shown below. Notice that the sprintf format nests all three types of quotes.

```
machine_id = sprintf('"`uname -n`"')
mycomputer = @machine_id
```

1.20 Syntax

Version 4 of gnuplot is much less sensitive than earlier versions to the order of keywords and suboptions. However, if you get error messages from specifying options that you think should work, please try rearranging them into the exact order listed by the documentation.

Options and any accompanying parameters are separated by spaces whereas lists and coordinates are separated by commas. Ranges are separated by colons and enclosed in brackets [], text and file names are enclosed in quotes, and a few miscellaneous things are enclosed in parentheses. Braces {} are used for a few special purposes.

Commas are used to separate coordinates on the 'set' commands Section 2.21.2 [arrow], page 75, Section 2.21.28 [key], page 100, and Section 2.21.29 [label], page 104; the list of variables being fitted (the list after the 'via' keyword on the 'fit' command); lists of discrete contours or the loop parameters which specify them on the Section 2.21.10 [cntrparam], page 83 command; the arguments of the 'set' commands Section 2.21.16 [dgrid3d], page 90, Section 2.21.17 [dummy], page 91, Section 2.21.27 [isosamples], page 100, Section 2.21.44 [offsets], page 114, Section 2.21.45 [origin], page 115, Section 2.21.56 [samples], page 128, Section 2.21.57 [size], page 129, 'time', and Section 2.21.74 [view], page 150; lists of tics or the loop parameters which specify them; the offsets for titles and axis labels; parametric functions to be used to calculate the x, y, and z coordinates on the 'plot', Section 2.17 [replot], page 71 and 'splot' commands; and the complete sets of keywords specifying individual plots (data sets or functions) on the 'plot', Section 2.17 [replot], page 71 and 'splot' commands.

Parentheses are used to delimit sets of explicit tics (as opposed to loop parameters) and to indicate computations in the Section 2.12.1.9 [using], page 61 filter of the 'fit', 'plot', Section 2.17 [replot], page 71 and 'splot' commands.

(Parentheses and commas are also used as usual in function notation.)

Square brackets are used to delimit ranges given in 'set', 'plot' or 'splot' commands.

Colons are used to separate extrema in 'range' specifications (whether they are given on 'set', 'plot' or 'splot' commands) and to separate entries in the Section 2.12.1.9 [using], page 61 filter of the 'plot', Section 2.17 [replot], page 71, 'splot' and 'fit' commands.

Semicolons are used to separate commands given on a single command line.

Braces are used in text to be specially processed by some terminals, like Section 2.21.50.6 [postscript], page 126. They are also used to denote complex numbers: {3,2} = 3 + 2i.

At present you should not embed \n inside {} when using the Section 2.21.50.6 [postscript], page 126 terminal.

The EEPIC, Imagen, Uniplex, LaTeX, and TPIC drivers allow a newline to be specified by \\ in a single-quoted string or \\\\ in a double-quoted string.

1.20.1 Quote Marks

Gnuplot uses three forms of quote marks for delimiting text strings, double-quote (ascii 34), single-quote (ascii 39), and backquote (ascii 96).

Filenames may be entered with either single- or double-quotes. In this manual the command examples generally single-quote filenames and double-quote other string tokens for clarity.

String constants and text strings used for labels, titles, or other plot elements may be enclosed in either single quotes or double quotes. Further processing of the quoted text depends on the choice of quote marks.

Backslash processing of special characters like \n (newline) and \345 (octal character code) is performed for double-quoted strings. In single-quoted strings, backslashes are just ordinary characters. To get a single-quote (ascii 39) in a single-quoted string, it has to be doubled. Thus the strings "d\" s' b\\" and 'd" s'' b\' are completely equivalent.

Text justification is the same for each line of a multi-line string. Thus the center-justified string

```
"This is the first line of text.\nThis is the second line."
```

will produce

```
This is the first line of text.
   This is the second line.
```

but

```
'This is the first line of text.\nThis is the second line.'
```

will produce

```
This is the first line of text.\nThis is the second line.
```

Enhanced text processing is performed for both double-quoted text and single-quoted text, but only by terminals supporting this mode. See 'enhanced text'.

Back-quotes are used to enclose system commands for substitution into the command line. See 'substitution'.

1.21 Time/Date data

'gnuplot' supports the use of time and/or date information as input data. This feature is activated by the commands 'set xdata time', 'set ydata time', etc.

Internally all times and dates are converted to the number of seconds from the year 2000. The command Section 2.21.67 [timefmt], page 147 defines the format for all inputs: data files, ranges, tics, label positions—in short, anything that accepts a data value must receive it in this format. Since only one input format can be in force at a given time, all time/date quantities being input at the same time must be presented in the same format. Thus if both x and y data in a file are time/date, they must be in the same format.

The conversion to and from seconds assumes Universal Time (which is the same as Greenwich Standard Time). There is no provision for changing the time zone or for daylight savings. If all your data refer to the same time zone (and are all either daylight or standard) you don't need to worry about these things. But if the absolute time is crucial for your application, you'll need to convert to UT yourself.

Commands like Section 2.21.87 [xrange], page 154 will re-interpret the integer according to Section 2.21.67 [timefmt], page 147. If you change Section 2.21.67 [timefmt], page 147, and then 'show' the quantity again, it will be displayed in the new Section 2.21.67 [timefmt], page 147. For that matter, if you give the deactivation command (like Section 2.21.83 [xdata], page 151), the quantity will be shown in its numerical form.

The command 'set format' defines the format that will be used for tic labels, whether or not the specified axis is time/date.

If time/date information is to be plotted from a file, the Section 2.12.1.9 [using], page 61 option _must_ be used on the 'plot' or 'splot' command. These commands simply use white space to separate columns, but white space may be embedded within the time/date string. If you use tabs as a separator, some trial-and-error may be necessary to discover how your system treats them.

The following example demonstrates time/date plotting.

Suppose the file "data" contains records like

```
03/21/95 10:00  6.02e23
```

This file can be plotted by

```
set xdata time
set timefmt "%m/%d/%y"
set xrange ["03/21/95":"03/22/95"]
set format x "%m/%d"
set timefmt "%m/%d/%y %H:%M"
plot "data" using 1:3
```

which will produce xtic labels that look like "03/21".

See the descriptions of each command for more details.

2 Commands

This section lists the commands acceptable to ‘gnuplot‘ in alphabetical order. Printed versions of this document contain all commands; on-line versions may not be complete. Indeed, on some systems there may be no commands at all listed under this heading.

Note that in most cases unambiguous abbreviations for command names and their options are permissible, i.e., "‘p f(x) w li‘" instead of "‘plot f(x) with lines‘".

In the syntax descriptions, braces ({}) denote optional arguments and a vertical bar (|) separates mutually exclusive choices.

2.1 cd

The Section 2.1 [cd], page 35 command changes the working directory.

Syntax:

```
cd '<directory-name>'
```

The directory name must be enclosed in quotes.

Examples:

```
cd 'subdir'
cd ".."
```

It is recommended for DOS and Windows users to use single-quotes—backslash [\] has special significance inside double-quotes and has to be escaped. For example,

```
cd "c:\newdata"
```

fails, but

```
cd 'c:\newdata'
cd "c:\\newdata"
```

works as expected.

2.2 call

The Section 2.2 [call], page 35 command is identical to the load command with one exception: you can have up to ten additional parameters to the command (delimited according to the standard parser rules) which can be substituted into the lines read from the file. As each line is read from the Section 2.2 [call], page 35ed input file, it is scanned for the sequence ‘$‘ (dollar-sign) followed by a digit (0–9). If found, the sequence is replaced by the corresponding parameter from the Section 2.2 [call], page 35 command line. If the parameter was specified as a string in the Section 2.2 [call], page 35 line, it is substituted without its enclosing quotes. Sequence ‘$#‘ is replaced by the number of passed parameters. ‘$‘ followed by any character will be that character; e.g. use ‘$$‘ to get a single ‘$‘. Providing more than ten parameters on the Section 2.2 [call], page 35 command line will cause an error. A parameter that was not provided substitutes as nothing. Files being Section 2.2 [call], page 35ed may themselves contain Section 2.2 [call], page 35 or ‘load‘ commands.

The Section 2.2 [call], page 35 command _must_ be the last command on a multi-command line.

Syntax:

```
call "<input-file>" <parameter-0> <parm-1> ... <parm-9>
```

The name of the input file must be enclosed in quotes, and it is recommended that parameters are similarly enclosed in quotes (future versions of gnuplot may treat quoted and unquoted arguments differently).

Example:

If the file 'calltest.gp' contains the line:

```
print "argc=$# p0=$0 p1=$1 p2=$2 p3=$3 p4=$4 p5=$5 p6=$6 p7=x$7x"
```

entering the command:

```
call 'calltest.gp' "abcd" 1.2 + "'quoted'" -- "$2"
```

will display:

```
argc=7 p0=abcd p1=1.2 p2=+ p3='quoted' p4=- p5=- p6=$2 p7=xx
```

NOTE: there is a clash in syntax with the datafile Section 2.12.1.9 [using], page 61 callback operator. Use '$$n' or 'column(n)' to access column n from a datafile inside a Section 2.2 [call], page 35ed datafile plot.

2.3 clear

The Section 2.3 [clear], page 36 command erases the current screen or output device as specified by Section 2.21.46 [output], page 115. This usually generates a formfeed on hardcopy devices. Use 'set terminal' to set the device type.

For some terminals Section 2.3 [clear], page 36 erases only the portion of the plotting surface defined by Section 2.21.57 [size], page 129, so for these it can be used in conjunction with Section 2.21.38 [multiplot], page 111 to create an inset.

Example:

```
set multiplot
plot sin(x)
set origin 0.5,0.5
set size 0.4,0.4
clear
plot cos(x)
unset multiplot
```

Please see Section 2.21.38 [multiplot], page 111, Section 2.21.57 [size], page 129, and Section 2.21.45 [origin], page 115 for details of these commands.

2.4 exit

The commands Section 2.4 [exit], page 36 and Section 2.15 [quit], page 71, as well as the END-OF-FILE character (usually Ctrl-D) terminate input from the current input stream: terminal session, pipe, and file input (pipe).

If input streams are nested (inherited ‘load‘ scripts), then reading will continue in the parent stream. When the top level stream is closed, the program itself will exit.

The command ‘exit gnuplot‘ will immediately and unconditionally cause gnuplot to exit even if the input stream is multiply nested. In this case any open output files may not be completed cleanly. Example of use:

```
bind "ctrl-x" "unset output; exit gnuplot"
```

See help for ‘batch/interactive‘ for more details.

2.5 fit

The ‘fit‘ command can fit a user-defined function to a set of data points (x,y) or (x,y,z), using an implementation of the nonlinear least-squares (NLLS) Marquardt-Levenberg algorithm. Any user-defined variable occurring in the function body may serve as a fit parameter, but the return type of the function must be real.

Syntax:

```
fit {[xrange] {[yrange]}} <function> '<datafile>'
    {datafile-modifiers}
    via '<parameter file>' | <var1>{,<var2>,...}
```

Ranges may be specified to temporarily limit the data which is to be fitted; any out-of-range data points are ignored. The syntax is

```
[{dummy_variable=}{<min>}{:<max>}],
```

analogous to ‘plot‘; see Section 2.12.5 [ranges], page 66.

<function> is any valid ‘gnuplot‘ expression, although it is usual to use a previously user-defined function of the form f(x) or f(x,y).

<datafile> is treated as in the ‘plot‘ command. All the Section 2.21.14 [datafile], page 86 modifiers (Section 2.12.1.9 [using], page 61, Section 2.12.1.3 [every], page 56,...) except Section 2.12.1.6 [smooth], page 57 and the deprecated Section 2.12.1.8 [thru], page 60 are applicable to ‘fit‘. See Section 2.21.14 [datafile], page 86.

The default data formats for fitting functions with a single independent variable, y=f(x), are {x:}y or x:y:s; those formats can be changed with the datafile Section 2.12.1.9 [using], page 61 qualifier. The third item (a column number or an expression), if present, is interpreted as the standard deviation of the corresponding y value and is used to compute a weight for the datum, 1/s**2. Otherwise, all data points are weighted equally, with a weight of one. Note that if you don’t specify a Section 2.12.1.9 [using], page 61 option at all, no y deviations are read from the datafile even if it does have a third column, so you’ll always get unit weights.

To fit a function with two independent variables, z=f(x,y), the required format is Section 2.12.1.9 [using], page 61 with four items, x:y:z:s. The complete format must be given—no default columns are assumed for a missing token. Weights for each data point are evaluated from ’s’ as above. If error estimates are not available, a constant value can be specified as a constant expression (see Section 2.12.1.9 [using], page 61), e.g., ‘using 1:2:3:(1)‘.

Multiple datasets may be simultaneously fit with functions of one independent variable by making y a 'pseudo-variable', e.g., the dataline number, and fitting as two independent variables. See Section 2.5.5 [multi-branch], page 43.

The ‘via‘ qualifier specifies which parameters are to be adjusted, either directly, or by referencing a parameter file.

Examples:

```
f(x) = a*x**2 + b*x + c
g(x,y) = a*x**2 + b*y**2 + c*x*y
FIT_LIMIT = 1e-6
fit f(x) 'measured.dat' via 'start.par'
fit f(x) 'measured.dat' using 3:($7-5) via 'start.par'
fit f(x) './data/trash.dat' using 1:2:3 via a, b, c
fit g(x,y) 'surface.dat' using 1:2:3:(1) via a, b, c
```

After each iteration step, detailed information about the current state of the fit is written to the display. The same information about the initial and final states is written to a log file, "fit.log". This file is always appended to, so as to not lose any previous fit history; it should be deleted or renamed as desired. By using the command ‘set fit logfile‘, the name of the log file can be changed.

If gnuplot was built with this option, and you activated it using ‘set fit errorvariables‘, the error for each fitted parameter will be stored in a variable named like the parameter, but with "_err" appended. Thus the errors can be used as input for further computations.

The fit may be interrupted by pressing Ctrl-C (any key but Ctrl-C under MSDOS and Atari Multitasking Systems). After the current iteration completes, you have the option to (1) stop the fit and accept the current parameter values, (2) continue the fit, (3) execute a ‘gnuplot‘ command as specified by the environment variable FIT_SCRIPT. The default for FIT_SCRIPT is Section 2.17 [replot], page 71, so if you had previously plotted both the data and the fitting function in one graph, you can display the current state of the fit.

Once ‘fit‘ has finished, the Section 2.27 [update], page 169 command may be used to store final values in a file for subsequent use as a parameter file. See Section 2.27 [update], page 169 for details.

2.5.1 adjustable parameters

There are two ways that ‘via‘ can specify the parameters to be adjusted, either directly on the command line or indirectly, by referencing a parameter file. The two use different means to set initial values.

Adjustable parameters can be specified by a comma-separated list of variable names after the ‘via‘ keyword. Any variable that is not already defined is created with an initial value of 1.0. However, the fit is more likely to converge rapidly if the variables have been previously declared with more appropriate starting values.

In a parameter file, each parameter to be varied and a corresponding initial value are specified, one per line, in the form

```
varname = value
```

Comments, marked by '#', and blank lines are permissible. The special form

```
varname = value       # FIXED
```

means that the variable is treated as a 'fixed parameter', initialized by the parameter file, but not adjusted by 'fit'. For clarity, it may be useful to designate variables as fixed parameters so that their values are reported by 'fit'. The keyword '# FIXED' has to appear in exactly this form.

2.5.2 short introduction

'fit' is used to find a set of parameters that 'best' fits your data to your user-defined function. The fit is judged on the basis of the sum of the squared differences or 'residuals' (SSR) between the input data points and the function values, evaluated at the same places. This quantity is often called 'chisquare' (i.e., the Greek letter chi, to the power of 2). The algorithm attempts to minimize SSR, or more precisely, WSSR, as the residuals are 'weighted' by the input data errors (or 1.0) before being squared; see 'fit error_estimates' for details.

That's why it is called 'least-squares fitting'. Let's look at an example to see what is meant by 'non-linear', but first we had better go over some terms. Here it is convenient to use z as the dependent variable for user-defined functions of either one independent variable, z=f(x), or two independent variables, z=f(x,y). A parameter is a user-defined variable that 'fit' will adjust, i.e., an unknown quantity in the function declaration. Linearity/nonlinearity refers to the relationship of the dependent variable, z, to the parameters which 'fit' is adjusting, not of z to the independent variables, x and/or y. (To be technical, the second {and higher} derivatives of the fitting function with respect to the parameters are zero for a linear least-squares problem).

For linear least-squares (LLS), the user-defined function will be a sum of simple functions, not involving any parameters, each multiplied by one parameter. NLLS handles more complicated functions in which parameters can be used in a large number of ways. An example that illustrates the difference between linear and nonlinear least-squares is the Fourier series. One member may be written as

```
z=a*sin(c*x) + b*cos(c*x).
```

If a and b are the unknown parameters and c is constant, then estimating values of the parameters is a linear least-squares problem. However, if c is an unknown parameter, the problem is nonlinear.

In the linear case, parameter values can be determined by comparatively simple linear algebra, in one direct step. However LLS is a special case which is also solved along with more general NLLS problems by the iterative procedure that 'gnuplot' uses. 'fit' attempts to find the minimum by doing a search. Each step (iteration) calculates WSSR with a new set of parameter values. The Marquardt-Levenberg algorithm selects the parameter values for the next iteration. The process continues until a preset criterion is met, either (1) the fit has "converged" (the relative change in WSSR is less than FIT_LIMIT), or (2) it reaches a preset iteration count limit, FIT_MAXITER (see Section 2.21.72 [variables], page 149). The fit may also be interrupted and subsequently halted from the keyboard (see 'fit').

Often the function to be fitted will be based on a model (or theory) that attempts to describe or predict the behaviour of the data. Then 'fit' can be used to find values for the free parameters of the model, to determine how well the data fits the model, and to estimate an error range for each parameter. See 'fit error_estimates'.

Alternatively, in curve-fitting, functions are selected independent of a model (on the basis of experience as to which are likely to describe the trend of the data with the desired resolution and a minimum number of parameters*functions.) The ‘fit‘ solution then provides an analytic representation of the curve.

However, if all you really want is a smooth curve through your data points, the Section 2.12.1.6 [smooth], page 57 option to ‘plot‘ may be what you've been looking for rather than ‘fit‘.

2.5.3 error estimates

In ‘fit‘, the term "error" is used in two different contexts, data error estimates and parameter error estimates.

Data error estimates are used to calculate the relative weight of each data point when determining the weighted sum of squared residuals, WSSR or chisquare. They can affect the parameter estimates, since they determine how much influence the deviation of each data point from the fitted function has on the final values. Some of the ‘fit‘ output information, including the parameter error estimates, is more meaningful if accurate data error estimates have been provided.

The 'statistical overview' describes some of the ‘fit‘ output and gives some background for the 'practical guidelines'.

2.5.3.1 statistical overview

The theory of non-linear least-squares (NLLS) is generally described in terms of a normal distribution of errors, that is, the input data is assumed to be a sample from a population having a given mean and a Gaussian (normal) distribution about the mean with a given standard deviation. For a sample of sufficiently large size, and knowing the population standard deviation, one can use the statistics of the chisquare distribution to describe a "goodness of fit" by looking at the variable often called "chisquare". Here, it is sufficient to say that a reduced chisquare (chisquare/degrees of freedom, where degrees of freedom is the number of datapoints less the number of parameters being fitted) of 1.0 is an indication that the weighted sum of squared deviations between the fitted function and the data points is the same as that expected for a random sample from a population characterized by the function with the current value of the parameters and the given standard deviations.

If the standard deviation for the population is not constant, as in counting statistics where variance = counts, then each point should be individually weighted when comparing the observed sum of deviations and the expected sum of deviations.

At the conclusion ‘fit‘ reports 'stdfit', the standard deviation of the fit, which is the rms of the residuals, and the variance of the residuals, also called 'reduced chisquare' when the data points are weighted. The number of degrees of freedom (the number of data points minus the number of fitted parameters) is used in these estimates because the parameters used in calculating the residuals of the datapoints were obtained from the same data.

To estimate confidence levels for the parameters, one can use the minimum chisquare obtained from the fit and chisquare statistics to determine the value of chisquare corresponding to the desired confidence level, but considerably more calculation is required to determine the combinations of parameters which produce such values.

Rather than determine confidence intervals, ‘fit‘ reports parameter error estimates which are readily obtained from the variance-covariance matrix after the final iteration. By convention, these estimates are called "standard errors" or "asymptotic standard errors", since they are calculated in the same way as the standard errors (standard deviation of each parameter) of a linear least-squares problem, even though the statistical conditions for designating the quantity calculated to be a standard deviation are not generally valid for the NLLS problem. The asymptotic standard errors are generally over-optimistic and should not be used for determining confidence levels, but are useful for qualitative purposes.

The final solution also produces a correlation matrix, which gives an indication of the correlation of parameters in the region of the solution; if one parameter is changed, increasing chisquare, does changing another compensate? The main diagonal elements, autocorrelation, are all 1; if all parameters were independent, all other elements would be nearly 0. Two variables which completely compensate each other would have an off-diagonal element of unit magnitude, with a sign depending on whether the relation is proportional or inversely proportional. The smaller the magnitudes of the off-diagonal elements, the closer the estimates of the standard deviation of each parameter would be to the asymptotic standard error.

2.5.3.2 practical guidelines

If you have a basis for assigning weights to each data point, doing so lets you make use of additional knowledge about your measurements, e.g., take into account that some points may be more reliable than others. That may affect the final values of the parameters.

Weighting the data provides a basis for interpreting the additional ‘fit‘ output after the last iteration. Even if you weight each point equally, estimating an average standard deviation rather than using a weight of 1 makes WSSR a dimensionless variable, as chisquare is by definition.

Each fit iteration will display information which can be used to evaluate the progress of the fit. (An '*' indicates that it did not find a smaller WSSR and is trying again.) The 'sum of squares of residuals', also called 'chisquare', is the WSSR between the data and your fitted function; ‘fit‘ has minimized that. At this stage, with weighted data, chisquare is expected to approach the number of degrees of freedom (data points minus parameters). The WSSR can be used to calculate the reduced chisquare (WSSR/ndf) or stdfit, the standard deviation of the fit, sqrt(WSSR/ndf). Both of these are reported for the final WSSR.

If the data are unweighted, stdfit is the rms value of the deviation of the data from the fitted function, in user units.

If you supplied valid data errors, the number of data points is large enough, and the model is correct, the reduced chisquare should be about unity. (For details, look up the 'chi-squared distribution' in your favourite statistics reference.) If so, there are additional tests, beyond the scope of this overview, for determining how well the model fits the data.

A reduced chisquare much larger than 1.0 may be due to incorrect data error estimates, data errors not normally distributed, systematic measurement errors, 'outliers', or an incorrect model function. A plot of the residuals, e.g., ‘plot 'datafile' using 1:($2-f($1))‘, may help to show any systematic trends. Plotting both the data points and the function may help to suggest another model.

Similarly, a reduced chisquare less than 1.0 indicates WSSR is less than that expected for a random sample from the function with normally distributed errors. The data error estimates may be too large, the statistical assumptions may not be justified, or the model function may be too general, fitting fluctuations in a particular sample in addition to the underlying trends. In the latter case, a simpler function may be more appropriate.

You'll have to get used to both 'fit' and the kind of problems you apply it to before you can relate the standard errors to some more practical estimates of parameter uncertainties or evaluate the significance of the correlation matrix.

Note that 'fit', in common with most NLLS implementations, minimizes the weighted sum of squared distances (y-f(x))**2. It does not provide any means to account for "errors" in the values of x, only in y. Also, any "outliers" (data points outside the normal distribution of the model) will have an exaggerated effect on the solution.

2.5.4 fit controlling

There are a number of 'gnuplot' variables that can be defined to affect 'fit'. Those which can be defined once 'gnuplot' is running are listed under 'control_variables' while those defined before starting 'gnuplot' are listed under 'environment_variables'.

2.5.4.1 control variables

The default epsilon limit (1e-5) may be changed by declaring a value for

```
FIT_LIMIT
```

When the sum of squared residuals changes between two iteration steps by a factor less than this number (epsilon), the fit is considered to have 'converged'.

The maximum number of iterations may be limited by declaring a value for

```
FIT_MAXITER
```

A value of 0 (or not defining it at all) means that there is no limit.

If you need even more control about the algorithm, and know the Marquardt-Levenberg algorithm well, there are some more variables to influence it. The startup value of 'lambda' is normally calculated automatically from the ML-matrix, but if you want to, you may provide your own one with

```
FIT_START_LAMBDA
```

Specifying FIT_START_LAMBDA as zero or less will re-enable the automatic selection. The variable

```
FIT_LAMBDA_FACTOR
```

gives the factor by which 'lambda' is increased or decreased whenever the chi-squared target function increased or decreased significantly. Setting FIT_LAMBDA_FACTOR to zero re-enables the default factor of 10.0.

Other variables with the FIT_ prefix may be added to 'fit', so it is safer not to use that prefix for user-defined variables.

The variables FIT_SKIP and FIT_INDEX were used by earlier releases of 'gnuplot' with a 'fit' patch called 'gnufit' and are no longer available. The datafile Section 2.12.1.3 [every], page 56 modifier provides the functionality of FIT_SKIP. FIT_INDEX was used for multi-branch fitting, but multi-branch fitting of one independent variable is now done as a pseudo-3D fit in which the second independent variable and Section 2.12.1.9 [using], page 61 are used to specify the branch. See Section 2.5.5 [multi-branch], page 43.

2.5.4.2 environment variables

The environment variables must be defined before ‘gnuplot‘ is executed; how to do so depends on your operating system.

`FIT_LOG`

changes the name (and/or path) of the file to which the fit log will be written from the default of "fit.log" in the working directory. The default value can be overwritten using the command ‘set fitlogfile‘.

`FIT_SCRIPT`

specifies a command that may be executed after an user interrupt. The default is Section 2.17 [replot], page 71, but a ‘plot‘ or ‘load‘ command may be useful to display a plot customized to highlight the progress of the fit.

2.5.5 multi-branch

In multi-branch fitting, multiple data sets can be simultaneously fit with functions of one independent variable having common parameters by minimizing the total WSSR. The function and parameters (branch) for each data set are selected by using a 'pseudo-variable', e.g., either the dataline number (a 'column' index of -1) or the datafile index (-2), as the second independent variable.

Example: Given two exponential decays of the form, z=f(x), each describing a different data set but having a common decay time, estimate the values of the parameters. If the datafile has the format x:z:s, then

```
f(x,y) = (y==0) ? a*exp(-x/tau) : b*exp(-x/tau)
fit f(x,y) 'datafile' using  1:-1:2:3  via a, b, tau
```

For a more complicated example, see the file "hexa.fnc" used by the "fit.dem" demo.

Appropriate weighting may be required since unit weights may cause one branch to predominate if there is a difference in the scale of the dependent variable. Fitting each branch separately, using the multi-branch solution as initial values, may give an indication as to the relative effect of each branch on the joint solution.

2.5.6 starting values

Nonlinear fitting is not guaranteed to converge to the global optimum (the solution with the smallest sum of squared residuals, SSR), and can get stuck at a local minimum. The routine has no way to determine that; it is up to you to judge whether this has happened.

‘fit‘ may, and often will get "lost" if started far from a solution, where SSR is large and changing slowly as the parameters are varied, or it may reach a numerically unstable region (e.g., too large a number causing a floating point overflow) which results in an "undefined value" message or ‘gnuplot‘ halting.

To improve the chances of finding the global optimum, you should set the starting values at least roughly in the vicinity of the solution, e.g., within an order of magnitude, if possible. The closer your starting values are to the solution, the less chance of stopping at another minimum. One way to find starting values is to plot data and the fitting function on the same graph and change parameter values and Section 2.17 [replot], page 71 until reasonable similarity is reached. The same plot is also useful to check whether the fit stopped at a minimum with a poor fit.

Of course, a reasonably good fit is not proof there is not a "better" fit (in either a statistical sense, characterized by an improved goodness-of-fit criterion, or a physical sense, with a solution more consistent with the model.) Depending on the problem, it may be desirable to ‘fit‘ with various sets of starting values, covering a reasonable range for each parameter.

2.5.7 tips

Here are some tips to keep in mind to get the most out of ‘fit‘. They’re not very organized, so you’ll have to read them several times until their essence has sunk in.

The two forms of the ‘via‘ argument to ‘fit‘ serve two largely distinct purposes. The ‘via "file"‘ form is best used for (possibly unattended) batch operation, where you just supply the startup values in a file and can later use Section 2.27 [update], page 169 to copy the results back into another (or the same) parameter file.

The ‘via var1, var2, ...‘ form is best used interactively, where the command history mechanism may be used to edit the list of parameters to be fitted or to supply new startup values for the next try. This is particularly useful for hard problems, where a direct fit to all parameters at once won’t work without good starting values. To find such, you can iterate several times, fitting only some of the parameters, until the values are close enough to the goal that the final fit to all parameters at once will work.

Make sure that there is no mutual dependency among parameters of the function you are fitting. For example, don’t try to fit a*exp(x+b), because a*exp(x+b)=a*exp(b)*exp(x). Instead, fit either a*exp(x) or exp(x+b).

A technical issue: the parameters must not be too different in magnitude. The larger the ratio of the largest and the smallest absolute parameter values, the slower the fit will converge. If the ratio is close to or above the inverse of the machine floating point precision, it may take next to forever to converge, or refuse to converge at all. You will have to adapt your function to avoid this, e.g., replace ’parameter’ by ’1e9*parameter’ in the function definition, and divide the starting value by 1e9.

If you can write your function as a linear combination of simple functions weighted by the parameters to be fitted, by all means do so. That helps a lot, because the problem is no longer nonlinear and should converge with only a small number of iterations, perhaps just one.

Some prescriptions for analysing data, given in practical experimentation courses, may have you first fit some functions to your data, perhaps in a multi-step process of accounting for several aspects of the underlying theory one by one, and then extract the information you really wanted from the fitting parameters of those functions. With ‘fit‘, this may often be done in one step by writing the model function directly in terms of the desired parameters. Transforming data can also quite often be avoided, though sometimes at the cost of a more difficult fit problem. If you think this contradicts the previous paragraph about simplifying the fit function, you are correct.

A "singular matrix" message indicates that this implementation of the Marquardt-Levenberg algorithm can't calculate parameter values for the next iteration. Try different starting values, writing the function in another form, or a simpler function.

Finally, a nice quote from the manual of another fitting package (fudgit), that kind of summarizes all these issues: "Nonlinear fitting is an art!"

2.6 help

The Section 2.6 [help], page 45 command displays on-line help. To specify information on a particular topic use the syntax:

```
help {<topic>}
```

If <topic> is not specified, a short message is printed about 'gnuplot'. After help for the requested topic is given, a menu of subtopics is given; help for a subtopic may be requested by typing its name, extending the help request. After that subtopic has been printed, the request may be extended again or you may go back one level to the previous topic. Eventually, the 'gnuplot' command line will return.

If a question mark (?) is given as the topic, the list of topics currently available is printed on the screen.

2.7 history

'history' command lists or saves previous entries in the history of the command line editing, or executes an entry.

Here you find 'usage by examples':

```
      history               # show the complete history
      history 5             # show last 5 entries in the history
      history quiet 5       # show last 5 entries without en-
try numbers
      history "hist.gp"     # write the complete his-
tory to file hist.gp
      history "hist.gp" append # append the complete his-
tory to file hist.gp
      history 10 "hist.gp"  # write last 10 commands to file hist.gp
      history 10 "|head -5 >>diary.gp" # write 5 history commands us-
ing pipe
      history ?load         # show all history entries start-
ing with "load"
      history ?"set c"      # like above, several words en-
closed in quotes
      hi !reread            # execute last entry start-
ing with "reread"
      hist !"set xr"        # like above, several words en-
closed in quotes
      hi !hi                # guess yourself :-))
```

On systems which support a popen function (Unix), the output of history can be piped through an external program by starting the file name with a '|', as one of the above examples demonstrates.

2.8 if

The Section 2.8 [if], page 46 command allows commands to be executed conditionally.

Syntax:

```
if (<condition>) <command-line> [; else if (<condi-
tion>) ...; else ...]
```

<condition> will be evaluated. If it is true (non-zero), then the command(s) of the <command-line> will be executed. If <condition> is false (zero), then the entire <command-line> is ignored until the next occurrence of 'else'. Note that use of ';' to allow multiple commands on the same line will _not_ end the conditionalized commands.

Examples:

```
pi=3
if (pi!=acos(-1)) print "?Fixing pi!"; pi=acos(-1); print pi
```

will display:

```
?Fixing pi!
3.14159265358979
```

but

```
if (1==2) print "Never see this"; print "Or this either"
```

will not display anything.

else:

```
v=0
v=v+1; if (v%2) print "2" ; else if (v%3) print "3"; else print "fred"
```

(repeat the last line repeatedly!)

See Section 2.18 [reread], page 72 for an example of how Section 2.8 [if], page 46 and Section 2.18 [reread], page 72 can be used together to perform a loop.

2.9 load

The 'load' command executes each line of the specified input file as if it had been typed in interactively. Files created by the Section 2.20 [save], page 73 command can later be 'load'ed. Any text file containing valid commands can be created and then executed by the 'load' command. Files being 'load'ed may themselves contain 'load' or Section 2.2 [call], page 35 commands. See 'comments' for information about comments in commands. To 'load' with arguments, see Section 2.2 [call], page 35.

The 'load' command _must_ be the last command on a multi-command line.

Syntax:

```
load "<input-file>"
```

The name of the input file must be enclosed in quotes.

The special filename "-" may be used to ‘load‘ commands from standard input. This allows a ‘gnuplot‘ command file to accept some commands from standard input. Please see help for ‘batch/interactive‘ for more details.

On some systems which support a popen function (Unix), the load file can be read from a pipe by starting the file name with a '<'.

Examples:

```
load 'work.gnu'
load "func.dat"
load "< loadfile_generator.sh"
```

The ‘load‘ command is performed implicitly on any file names given as arguments to ‘gnuplot‘. These are loaded in the order specified, and then ‘gnuplot‘ exits.

2.10 lower

Syntax:

```
lower {x11_plot_window_nb}
```

The Section 2.10 [lower], page 47 command lowers (opposite to Section 2.16 [raise], page 71) plot window(s) associated with the interactive terminal of your gnuplot session, i.e. ‘pm‘, ‘win‘ or ‘x11‘. It puts the plot window to bottom in the z-order windows stack of the window manager of your desktop.

As ‘x11‘ supports multiple plot windows, then by default it lowers these windows in descending order of most recently created on top to the least recently created on bottom. If a plot number is supplied as an optional parameter, only the associated plot window will be lowered if it exists.

The optional parameter is ignored for single plot-window terminals, i.e. ‘pm‘ and ‘win‘.

2.11 pause

The Section 2.11 [pause], page 47 command displays any text associated with the command and then waits a specified amount of time or until the carriage return is pressed. Section 2.11 [pause], page 47 is especially useful in conjunction with ‘load‘ files.

Syntax:

```
pause <time> {"<string>"}
pause mouse {<endcondition>}{, <endcondition>} {"<string>"}
```

<time> may be any constant or expression. Choosing -1 will wait until a carriage return is hit, zero (0) won't pause at all, and a positive number will wait the specified number of seconds. The time is rounded to an integer number of seconds if subsecond time resolution is not supported by the given platform. ‘pause 0‘ is synonymous with ‘print‘.

If the current terminal supports mousing, then ‘pause mouse‘ will terminate on either a mouse click or on ctrl-C. For all other terminals, or if mousing is not active, ‘pause mouse‘ is equivalent to ‘pause -1‘.

If one or more end conditions are given after ‘pause mouse‘, then any one of the conditions will terminate the pause. The possible end conditions are ‘keypress‘, ‘button1‘, ‘button2‘, ‘button3‘, and ‘any‘. If the pause terminates on a keypress, then the ascii value of the key pressed is returned in MOUSE_KEY. The character itself is returned as a one character string in MOUSE_CHAR.

In all cases the coordinates of the mouse are returned in variables MOUSE_X, MOUSE_Y, MOUSE_X2, MOUSE_Y2. See Section 2.21.72 [variables], page 149.

Note: Since Section 2.11 [pause], page 47 communicates with the operating system rather than the graphics, it may behave differently with different device drivers (depending upon how text and graphics are mixed).

Examples:

```
pause -1    # Wait until a carriage return is hit
pause 3     # Wait three seconds
pause -1  "Hit return to continue"
pause 10  "Isn't this pretty?  It's a cubic spline."
pause mouse "Click any mouse button on selected data point"
pause mouse keypress "Type a letter from A-F in the ac-
tive window"
pause mouse button1,keypress
pause mouse any "Any key or button will terminate"
```

The variant "pause mouse key" will resume after any keypress in the active plot window. If you want to wait for a particular key to be pressed, you can use a reread loop such as:

```
printf "I will resume af-
ter you hit the Tab key in the plot window"
load "wait_for_tab"
```

File "wait_for_tab" contains the lines

```
pause mouse key
if (MOUSE_KEY != 9) reread
```

2.12 plot

‘plot‘ is the primary command for drawing plots with ‘gnuplot‘. It creates plots of functions and data in many, many ways. ‘plot‘ is used to draw 2-d functions and data; ‘splot‘ draws 2-d projections of 3-d surfaces and data. ‘plot‘ and ‘splot‘ contain many common features; see ‘splot‘ for differences. Note specifically that although the ‘binary <binary list>‘ variation does work for both ‘plot‘ and ‘splot‘, there are small differences between these modes. Furthermore, ‘plot‘’s ‘axes‘ option does not exist for ‘splot‘.

Syntax:

```
plot {<ranges>}
     {<function> | {"<datafile>" {datafile-modifiers}}}
     {axes <axes>} {<title-spec>} {with <style>}
     {, {definitions,} <function> ...}
```

where either a <function> or the name of a data file enclosed in quotes is supplied. A function is a mathematical expression or a pair of mathematical expressions in parametric mode. The expressions may be defined completely or in part earlier in the stream of ‘gnuplot‘ commands (see ‘user-defined‘).

It is also possible to define functions and parameters on the ‘plot‘ command itself. This is done merely by isolating them from other items with commas.

There are four possible sets of axes available; the keyword <axes> is used to select the axes for which a particular line should be scaled. ‘x1y1‘ refers to the axes on the bottom and left; ‘x2y2‘ to those on the top and right; ‘x1y2‘ to those on the bottom and right; and ‘x2y1‘ to those on the top and left. Ranges specified on the ‘plot‘ command apply only to the first set of axes (bottom left).

Examples:

```
plot sin(x)
plot f(x) = sin(x*a), a = .2, f(x), a = .4, f(x)
plot [t=1:10] [-pi:pi*2] tan(t), \
     "data.1" using (tan($2)):($3/$4) smooth csplines \
              axes x1y2 notitle with lines 5
```

See also ‘show plot‘.

2.12.1 data

Discrete data contained in a file can be displayed by specifying the name of the data file (enclosed in single or double quotes) on the ‘plot‘ command line.

Syntax:

```
plot '<file_name>' {binary <binary list>}
                   {matrix}
                   {index <index list>}
                   {every <every list>}
                   {thru <thru expression>}
                   {using <using list>}
                   {smooth <option>}
```

The modifiers ‘binary‘, Section 2.12.1.5 [index], page 57, Section 2.12.1.3 [every], page 56, Section 2.12.1.8 [thru], page 60, Section 2.12.1.9 [using], page 61, and Section 2.12.1.6 [smooth], page 57 are discussed separately. In brief, ‘binary‘ allows data entry from a binary file (default is ASCII), Section 2.12.1.5 [index], page 57 selects which data sets in a multi-data-set file are to be plotted, Section 2.12.1.3 [every], page 56 specifies which points within a single data set are to be plotted, Section 2.12.1.9 [using], page 61 determines how the columns within a single record are to be interpreted (Section 2.12.1.8 [thru], page 60 is a special case of Section 2.12.1.9 [using], page 61), and Section 2.12.1.6 [smooth], page 57 allows for simple interpolation and approximation. (‘splot‘ has a similar syntax, but does not support the Section 2.12.1.6 [smooth], page 57 and Section 2.12.1.8 [thru], page 60 options.)

ASCII DATA FILES:

Data files should contain at least one data point per record (Section 2.12.1.9 [using], page 61 can select one data point from the record). Records beginning with '#' (and also with '¡ on VMS) will be treated as comments and ignored. Each data point represents an (x,y) pair. For 'plot's with error bars or error bars with lines (see Section 2.12.2 [errorbars], page 64 or Section 2.12.3 [errorlines], page 64), each data point is (x,y,ydelta), (x,y,ylow,yhigh), (x,y,xdelta), (x,y,xlow,xhigh), or (x,y,xlow,xhigh,ylow,yhigh).

In all cases, the numbers of each record of a data file must be separated by white space (one or more blanks or tabs) unless a format specifier is provided by the Section 2.12.1.9 [using], page 61 option. This white space divides each record into columns. However, whitespace inside a pair of double quotes is ignored when counting columns, so the following datafile line has three columns:

```
1.0 "second column" 3.0
```

Data may be written in exponential format with the exponent preceded by the letter e, E, d, D, q, or Q.

Only one column (the y value) need be provided. If x is omitted, 'gnuplot' provides integer values starting at 0.

In datafiles, blank records (records with no characters other than blanks and a newline and/or carriage return) are significant—pairs of blank records separate Section 2.12.1.5 [index], page 57es (see Section 2.12.1.5 [index], page 57). Data separated by double blank records are treated as if they were in separate data files.

Single blank records designate discontinuities in a 'plot'; no line will join points separated by a blank records (if they are plotted with a line style).

If autoscaling has been enabled (Section 2.21.3 [autoscale], page 77), the axes are automatically extended to include all datapoints, with a whole number of tic marks if tics are being drawn. This has two consequences: i) For 'splot', the corner of the surface may not coincide with the corner of the base. In this case, no vertical line is drawn. ii) When plotting data with the same x range on a dual-axis graph, the x coordinates may not coincide if the x2tics are not being drawn. This is because the x axis has been autoextended to a whole number of tics, but the x2 axis has not. The following example illustrates the problem:

```
reset; plot '-', '-' axes x2y1
1 1
19 19
e
1 1
19 19
e
```

To avoid this, you can use the 'fixmin'/'fixmax' feature of the Section 2.21.3 [autoscale], page 77 command, which turns off the automatic extension of the axis range upto the next tic mark.

BINARY DATA FILES:

Gnuplot can read binary data files. However, adequate information about details of the file format must be given on the command line or extracted from the file itself for a supported binary ‘filetype‘. In particular, there are two structures for binary files, a matrix binary format and a general binary format.

The matrix binary format contains a two dimensional array of 32 bit IEEE float values with an additional column and row of coordinate values. As with ASCII matrix, in the using list, repetition of the coordinate row constitutes column 1, repetition of the coordinate column constitutes column 2, and the array of values constitutes column 3.

The general binary format contains an arbitrary number of columns fow which information must be specified at the command line. For example, ‘array‘, ‘record‘, ‘format‘ and Section 2.12.1.9 [using], page 61 can indicate the size, format and dimension of data. There are a variety of useful commands for skipping file headers, changing endianess. There are a set of commands for positioning and translating data since often coordinates are not part of the file when uniform sampling is inherent in the data. Different from matrix binary or ASCII, general binary does not treat the generated columns as 1, 2 or 3 in the using list. Rather, column 1 begins with column 1 of the file, or as specified in the ‘format‘ list.

There are global default settings for the various binary options which may be set using the same syntax as the options when used as part of the ‘(s)plot <filename> binary ...‘ command. This syntax is ‘set datafile binary ...‘. The general rule is that common command-line specified parameters override file-extracted parameters which override default parameters.

Matrix binary is the default binary format when no keywords specific to general binary are given, i.e., ‘array‘, ‘record‘, ‘format‘, ‘filetype‘.

General binary data can be entered at the command line via the special file name ’-’. However, this is intended for use through a pipe where programs can exchange binary data, not for keyboards. There is no "end of record" character for binary data. Gnuplot continues reading from a pipe until it has read the number of points declared in the ‘array‘ qualifier.

See ‘datafile binary‘ for more details.

2.12.1.1 binary

The ‘binary‘ keyword allows a data file to be binary as opposed to ASCII. There are two formats for binary–matrix binary and general binary. Matrix binary is a fixed format in which data appears in a 2D array with an extra row and column for coordinate values. General binary is a flexible format for which details about the file must be given at the command line.

See ‘binary matrix‘ or ‘binary general‘ for more details.

2.12.1.2 binary general

General binary data in which format information is not necessarily part of the file can be read by giving further details about the file format at the command line. Although the syntax is slightly arcane to the casual user, general binary is particularly useful for application programs using gnuplot and sending large amounts of data.

Syntax:

```
plot '<file_name>' {binary <binary list>} ...
splot '<file_name>' {binary <binary list>} ...
```

General binary format is activated by keywords in <binary list> pertaining to information about file structure, i.e., 'array', 'record', 'format' or 'filetype'. Otherwise, matrix binary format is assumed. (See 'binary matrix' for more details.)

There are some standard file types that may be read for which details about the binary format may be extracted automatically. (Type 'show datafile binary' at the command line for a list.) Otherwise, details must be specified at the command line or set in the defaults. Keywords are described below.

The keyword 'filetype' in <binary list> controls the routine used to read the file, i.e., the format of the data. For a list of the supported file types, type 'show datafile binary filetypes'. If no file type is given, the rule is that "traditional" gnuplot binary is assumed for 'splot' if the 'binary' keyword stands alone. In all other circumstances, for 'plot' or when one of the <binary list> keywords appears, a raw binary file is assumed whereby the keywords specify the binary format.

General binary data files fall into two basic classes, and some files may be of both classes depending upon how they are treated. There is that class for which uniform sampling is assumed and point coordinates must be generated. This is the class for which full control via the <binary list> keywords applies. For this class, the settings precedence is that command line parameters override in-file parameters, which override default settings. The other class is that set of files for which coordinate information is contained within the file or there is possibly a non-uniform sampling such as gnuplot binary.

Other than for the unique data files such as gnuplot binary, one should think of binary data as conceptually the same as ASCII data. Each point has columns of information which are selected via the '<using list>' associated with Section 2.12.1.9 [using], page 61. When no 'format' string is specified, gnuplot will retrieve a number of binary variables equal to the largest column given in the '<using list>'. For example, 'using 1:3' will result in three columns being read, of which the second will be ignored. There are default using lists based upon the typical number of parameters associated with a certain plot type. For example, 'with image' has a default of 'using 1', while 'with rgbimage' has a default of 'using 1:2:3'. Note that the special characters for Section 2.12.1.9 [using], page 61 representing point/line/index generally should not be used for binary data. There are keywords in <binary list> that control this.

— ARRAY —

Describes the sampling array dimensions associated with the binary file. The coordinates will be generated by gnuplot. A number must be specified for each dimension, thereby calling out the size of the array. For example, 'array=10x20' means the underlying sampling structure is two-dimensional with 10 points along the first (x) dimension and 20 points along the second (y) dimension. A special "number", 'Inf', can be used to indicate that data should be read until the end of file. A comma can be used to separate the dimensions for multiple records. For example, 'array=25,35' indicates there are two one-dimensional records within the file. The comma behavior applies to the remaining keywords in this list for which it makes sense to be associated with individual records.

Currently, syntax allows for up to three-dimensional arrays. However, no conventions have yet been made for handling three-dimensional coordinates.

— RECORD —

This keyword serves the same function as 'array', having the same syntax. However, 'record' causes gnuplot to not generate coordinate information. This is for the case where such information may be included in one of the columns of the binary data file.

— FORMAT —

The default binary format is a float. For more flexibility, the format can include details about variable sizes. For example, 'format="%uchar%int%float"' associates an unsigned character with the first using column, an int with the second column and a float with the third column. If the number of size specifications is less than the greatest column number, the size is implicitly taken to be similar to the last given variable size.

Furthermore, the format specification can include "discarded" terms via the '*' character. For example, to skip the middle column of the previous example, one could write 'format="%uchar%*int%float"' and gnuplot will discard the middle integer. To list variable sizes, type 'show datafile binary datasizes'. There are a group of names that are machine dependent along with their sizes in bytes for the particular compilation. There is also a group of names which attempt to be machine independent.

— ENDIAN —

Often the endianess of binary data in the file does not agree with the endianess used by the platform on which gnuplot is running. Several words can direct gnuplot how to arrange bytes. For example 'endian=little' means treat the binary file as having byte significance from least to greatest. The options are

```
             little:  least significant to greatest significance
                big:  greatest significance to least significance
            default:  assume file endianess is the same as compiler
        swap (swab):  Interchange the significance.  (If things
                      don't look right, try this.)
```

Gnuplot can support "middle" ("pdp") endian if it is compiled with that option.

— FILETYPE —

For some standard binary file formats gnuplot can extract all the necessary information from the file in question. As an example, "format=edf" will read ESRF Header File format files. For a list of the currently supported file formats, type 'show datafile binary filetypes'.

There is a special file type called 'auto' for which gnuplot will check if the binary file's extension is a quasi-standard extension for a supported format.

Command line keywords may be used to override settings extracted from the file. The settings from the file override any defaults. (See 'set datafile binary' for details.)

— AVS —

'avs' is one of the automatically recognized binary file types for images. AVS is an extremely simple format, suitable mostly for streaming between applications. It consists of 2 longs (xwidth, ywidth) followed by a stream of pixels, each with four bytes of information alpha/red/green/blue.

— EDF —

'edf' is one of the automatically recognized binary file types for images. EDF stands for ESRF Data Format, and it supports both edf and ehf formats (the latter means ESRF Header Format). More information on specifications can be found at `http://www.esrf.fr/computing/expg/subgroups/general/format/Format.html`

See also ‘binary‘.

— KEYWORDS —

The following keywords apply only when generating coordinates. That is, when the keyword ‘array‘ is used.

— SCAN —

A great deal of confusion can arise concerning the relationship between how gnuplot scans a binary file and the dimensions seen on the plot. To lessen the confusion, conceptually think of gnuplot _always_ scanning the binary file point/line/plane or fast/medium/slow. Then this keyword is used to tell gnuplot how to map this scanning convention to the Cartesian convention shown in plots, i.e., x/y/z. The qualifier for scan is a two or three letter code representing where point is assigned (first letter), line is assigned (second letter), and plane is assigned (third letter). For example, ‘scan=yx‘ means the fastest, point-by-point, increment should be mapped along the Cartesian y dimension and the middle, line-by-line, increment should be mapped along the x dimension.

When the plotting mode is ‘plot‘, the qualifier code can include the two letters x and y. For ‘splot‘, it can include the three letters x, y and z.

There is nothing restricting the inherent mapping from point/line/plane to apply only to Cartesian coordinates. For this reason there are cylindrical coordinate synonyms for the qualifier codes where t (theta), r and z are analogous to the x, y and z of Cartesian coordinates.

— TRANSPOSE —

Shorthand notation for ‘scan=yx‘ or ‘scan=yxz‘.

— DX, DY, DZ —

When gnuplot generates coordinates, it uses the spacing described by these keywords. For example ‘dx=10 dy=20‘ would mean space samples along the x dimension by 10 and space samples along the y dimension by 20. ‘dy‘ cannot appear if ‘dx‘ does not appear. Similarly, ‘dz‘ cannot appear if ‘dy‘ does not appear. If the underlying dimensions are greater than the keywords specified, the spacing of the highest dimension given is extended to the other dimensions. For example, if an image is being read from a file and only ‘dx=3.5‘ is given gnuplot uses a delta x and delta y of 3.5.

The following keywords also apply only when generating coordinates. However they may also be used with matrix binary files.

— FLIPX, FLIPY, FLIPZ —

Sometimes the scanning directions in a binary datafile are not consistent with that assumed by gnuplot. These keywords can flip the scanning direction along dimensions x, y, z.

— ORIGIN —

When gnuplot generates coordinates based upon transposition and flip, it attempts to always position the lower left point in the array at the origin, i.e., the data lies in the first quadrant of a Cartesian system after transpose and flip.

To position the array somewhere else on the graph, the Section 2.21.45 [origin], page 115 keyword directs gnuplot to position the lower left point of the array at a point specified by a tuple. The tuple should be a double for 'plot' and a triple for 'splot'. For example, 'origin=(100,100),(100,200)' is for two records in the file and intended for plotting in two dimensions. A second example, 'origin=(0,0,3.5)', is for plotting in three dimensions.

— CENTER —

Similar to Section 2.21.45 [origin], page 115, this keyword will position the array such that its center lies at the point given by the tuple. For example, 'center=(0,0)'. Center does not apply when the size of the array is 'Inf'.

— ROTATE —

The transpose and flip commands provide some flexibility in generating and orienting coordinates. However, for full degrees of freedom, it is possible to apply a rotational vector described by a rotational angle in two dimensions.

The 'rotate' keyword applies to the two-dimensional plane, whether it be 'plot' or 'splot'. The rotation is done with respect to the positive angle of the Cartesian plane.

The angle can be expressed in radians, radians as a multiple of pi, or degrees. For example, 'rotate=1.5708', 'rotate=0.5pi' and 'rotate=90deg' are equivalent.

If Section 2.21.45 [origin], page 115 is specified, the rotation is done about the lower left sample point before translation. Otherwise, the rotation is done about the array 'center'.

— PERPENDICULAR —

For 'splot', the concept of a rotational vector is implemented by a triple representing the vector to be oriented normal to the two-dimensional x-y plane. Naturally, the default is (0,0,1). Thus specifying both rotate and perpendicular together can orient data myriad ways in three-space.

The two-dimensional rotation is done first, followed by the three-dimensional rotation. That is, if R' is the rotational 2 x 2 matrix described by an angle, and P is the 3 x 3 matrix projecting (0,0,1) to (xp,yp,zp), let R be constructed from R' at the upper left sub-matrix, 1 at element 3,3 and zeros elsewhere. Then the matrix formula for translating data is v' = P R v, where v is the 3 x 1 vector of data extracted from the data file. In cases where the data of the file is inherently not three-dimensional, logical rules are used to place the data in three-space. (E.g., usually setting the z-dimension value to zero and placing 2D data in the x-y plane.)

— BINARY_EXAMPLES —

Examples:

```
      # Selects two float values (second one im-
plicit) with a float value
      # discarded between them for an indefinite length of 1D data.
      plot '<file_name>' binary format="%float%*float" us-
ing 1:2 with lines

      # The data file header contains all details neces-
sary for creating
      # coordinates from an EDF file.
      plot '<file_name>' binary filetype=edf with image
```

```
      plot '<file_name>.edf' binary filetype=auto with image

      # Selects three unsigned characters for compo-
nents of a raw RGB image
      # and flips the y-dimension so that typical image orienta-
tion (start
      # at top left corner) translates to the Cartesian plane.  Pixel
      # spacing is given and there are two im-
ages in the file.  One of them
      # is translated via origin.
      plot '<file_name>' binary array=512x1024,1024x512 for-
mat='%uchar' \
           dx=2,1 dy=1,2 ori-
gin=(0,0),(1024,1024) flipy u 1:2:3 w rgbimage

      # Four separate records in which the coordinates are part of the
      # data file.  The file was created with a endianess differ-
ent from
      # the system on which gnuplot is running.
      splot '<file_name>' binary record=30,30,29,26 en-
dian=swap u 1:2:3
```

See also 'binary matrix'.

2.12.1.3 every

The Section 2.12.1.3 [every], page 56 keyword allows a periodic sampling of a data set to be plotted.

In the discussion a "point" is a datum defined by a single record in the file; "block" here will mean the same thing as "datablock" (see 'glossary').

Syntax:

```
      plot 'file' every {<point_incr>}
                          {:{<block_incr>}
                            {:{<start_point>}
                              {:{<start_block>}
                                {:{<end_point>}
                                  {:<end_block>}}}}}
```

The data points to be plotted are selected according to a loop from <'start_point'> to <'end_point'> with increment <'point_incr'> and the blocks according to a loop from <'start_block'> to <'end_block'> with increment <'block_incr'>.

The first datum in each block is numbered '0', as is the first block in the file.

Note that records containing unplottable information are counted.

Any of the numbers can be omitted; the increments default to unity, the start values to the first point or block, and the end values to the last point or block. If Section 2.12.1.3 [every], page 56 is not specified, all points in all lines are plotted.

Examples:

```
      every :::3::3     # selects just the fourth block ('0' is first)
      every :::::9      # selects the first 10 blocks
      every 2:2         # selects every other point in ev-
ery other block
      every ::5::15     # selects points 5 through 15 in each block
```

See simple plot demos (simple.dem) , Non-parametric splot demos , and Parametric splot demos .

2.12.1.4 example datafile

This example plots the data in the file "population.dat" and a theoretical curve:

```
pop(x) = 103*exp((1965-x)/10)
plot [1960:1990] 'population.dat', pop(x)
```

The file "population.dat" might contain:

```
# Gnu population in Antarctica since 1965
   1965   103
   1970   55
   1975   34
   1980   24
   1985   10
```

2.12.1.5 index

The Section 2.12.1.5 [index], page 57 keyword allows only some of the data sets in a multi-data-set file to be plotted.

Syntax:

```
plot 'file' index <m>{{:<n>}:<p>}
```

Data sets are separated by pairs of blank records. 'index <m>' selects only set <m>; 'index <m>:<n>' selects sets in the range <m> to <n>; and 'index <m>:<n>:<p>' selects indices <m>, <m>+<p>, <m>+2<p>, etc., but stopping at <n>. Following C indexing, the index 0 is assigned to the first data set in the file. Specifying too large an index results in an error message. If Section 2.12.1.5 [index], page 57 is not specified, all sets are plotted as a single data set.

Example:

```
plot 'file' index 4:5
```

splot with indices demo.

2.12.1.6 smooth

'gnuplot' includes a few general-purpose routines for interpolation and approximation of data; these are grouped under the Section 2.12.1.6 [smooth], page 57 option. More sophisticated data processing may be performed by preprocessing the data externally or by using 'fit' with an appropriate model.

Syntax:

```
        smooth {unique | frequency | csplines | ac-
  splines | bezier | sbezier}
```

'unique' and 'frequency' plot the data after making them monotonic. Each of the other routines uses the data to determine the coefficients of a continuous curve between the endpoints of the data. This curve is then plotted in the same manner as a function, that is, by finding its value at uniform intervals along the abscissa (see Section 2.21.56 [samples], page 128) and connecting these points with straight line segments (if a line style is chosen).

If Section 2.21.3 [autoscale], page 77 is in effect, the ranges will be computed such that the plotted curve lies within the borders of the graph.

If Section 2.21.3 [autoscale], page 77 is not in effect, and the smooth option is either 'acspline' or 'cspline', the sampling of the generated curve is done across the intersection of the x range covered by the input data and the fixed abscissa range as defined by Section 2.21.87 [xrange], page 154.

If too few points are available to allow the selected option to be applied, an error message is produced. The minimum number is one for 'unique' and 'frequency', four for 'acsplines', and three for the others.

The Section 2.12.1.6 [smooth], page 57 options have no effect on function plots.

— ACSPLINES —

The 'acsplines' option approximates the data with a "natural smoothing spline". After the data are made monotonic in x (see 'smooth unique'), a curve is piecewise constructed from segments of cubic polynomials whose coefficients are found by the weighting the data points; the weights are taken from the third column in the data file. That default can be modified by the third entry in the Section 2.12.1.9 [using], page 61 list, e.g.,

```
plot 'data-file' using 1:2:(1.0) smooth acsplines
```

Qualitatively, the absolute magnitude of the weights determines the number of segments used to construct the curve. If the weights are large, the effect of each datum is large and the curve approaches that produced by connecting consecutive points with natural cubic splines. If the weights are small, the curve is composed of fewer segments and thus is smoother; the limiting case is the single segment produced by a weighted linear least squares fit to all the data. The smoothing weight can be expressed in terms of errors as a statistical weight for a point divided by a "smoothing factor" for the curve so that (standard) errors in the file can be used as smoothing weights.

Example:

```
sw(x,S)=1/(x*x*S)
plot 'data_file' using 1:2:(sw($3,100)) smooth acsplines
```

— BEZIER —

The 'bezier' option approximates the data with a Bezier curve of degree n (the number of data points) that connects the endpoints.

— CSPLINES —

The ‘csplines‘ option connects consecutive points by natural cubic splines after rendering the data monotonic (see ‘smooth unique‘).

— SBEZIER —

The ‘sbezier‘ option first renders the data monotonic (‘unique‘) and then applies the ‘bezier‘ algorithm.

— UNIQUE —

The ‘unique‘ option makes the data monotonic in x; points with the same x-value are replaced by a single point having the average y-value. The resulting points are then connected by straight line segments. demos

— FREQUENCY —

The ‘frequency‘ option makes the data monotonic in x; points with the same x-value are replaced by a single point having the summed y-values. The resulting points are then connected by straight line segments.

2.12.1.7 special-filenames

A special filename of ‘’-’‘ specifies that the data are inline; i.e., they follow the command. Only the data follow the command; ‘plot‘ options like filters, titles, and line styles remain on the ‘plot‘ command line. This is similar to `<<` in unix shell script, and $DECK in VMS DCL. The data are entered as though they are being read from a file, one data point per record. The letter `"e"` at the start of the first column terminates data entry. The Section 2.12.1.9 [using], page 61 option can be applied to these data—using it to filter them through a function might make sense, but selecting columns probably doesn’t!

‘’-’‘ is intended for situations where it is useful to have data and commands together, e.g., when ‘gnuplot‘ is run as a sub-process of some front-end application. Some of the demos, for example, might use this feature. While ‘plot‘ options such as Section 2.12.1.5 [index], page 57 and Section 2.12.1.3 [every], page 56 are recognized, their use forces you to enter data that won’t be used. For example, while

```
plot '-' index 0, '-' index 1
2
4
6


10
12
14
e
2
4
6


10
12
14
```

```
e
```

does indeed work,

```
plot '-', '-'
2
4
6
e
10
12
14
e
```

is a lot easier to type.

If you use ''-'' with Section 2.17 [replot], page 71, you may need to enter the data more than once (see Section 2.17 [replot], page 71).

A blank filename (") specifies that the previous filename should be reused. This can be useful with things like

```
plot 'a/very/long/filename' using 1:2, '' using 1:3, '' us-
ing 1:4
```

(If you use both ''-'' and '"' on the same 'plot' command, you'll need to have two sets of inline data, as in the example above.)

On some computer systems with a popen function (Unix), the datafile can be piped through a shell command by starting the file name with a '<'. For example,

```
pop(x) = 103*exp(-x/10)
plot "< awk '{print $1-1965, $2}' population.dat", pop(x)
```

would plot the same information as the first population example but with years since 1965 as the x axis. If you want to execute this example, you have to delete all comments from the data file above or substitute the following command for the first part of the command above (the part up to the comma):

```
plot "< awk '$0 !~ /^#/ {print $1-1965, $2}' population.dat"
```

While this approach is most flexible, it is possible to achieve simple filtering with the Section 2.12.1.9 [using], page 61 or Section 2.12.1.8 [thru], page 60 keywords.

2.12.1.8 thru

The Section 2.12.1.8 [thru], page 60 function is provided for backward compatibility.

Syntax:

```
plot 'file' thru f(x)
```

It is equivalent to:

```
plot 'file' using 1:(f($2))
```

While the latter appears more complex, it is much more flexible. The more natural

```
plot 'file' thru f(y)
```

also works (i.e. you can use y as the dummy variable).

Section 2.12.1.8 [thru], page 60 is parsed for 'splot' and 'fit' but has no effect.

2.12.1.9 using

The most common datafile modifier is Section 2.12.1.9 [using], page 61.

Syntax:

```
plot 'file' using {<entry> {:<entry> {:<entry> ...}}} {'format'}
```

If a format is specified, each datafile record is read using the C library's 'scanf' function, with the specified format string. Otherwise the record is read and broken into columns at spaces or tabs. A format cannot be specified if time-format data is being used (this must be done by 'set data time').

The resulting array of data is then sorted into columns according to the entries. Each `<entry>` may be a simple column number, which selects the datum, an expression enclosed in parentheses, or empty. The expression can use $1 to access the first item read, $2 for the second item, and so on. It can also use 'column(x)' and 'valid(x)' where x is an arbitrary expression resulting in an integer. 'column(x)' returns the x'th datum; 'valid(x)' tests that the datum in the x'th column is a valid number. A column number of 0 generates a number increasing (from zero) with each point, and is reset upon encountering two blank records. A column number of -1 gives the dataline number, which starts at 0, increments at single blank records, and is reset at double blank records. A column number of -2 gives the index number, which is incremented only when two blank records are found. An empty `<entry>` will default to its order in the list of entries. For example, 'using ::4' is interpreted as 'using 1:2:4'.

N.B.—the Section 2.2 [call], page 35 command also uses $'s as a special character. See Section 2.2 [call], page 35 for details about how to include a column number in a Section 2.2 [call], page 35 argument list.

If the Section 2.12.1.9 [using], page 61 list has but a single entry, that `<entry>` will be used for y and the data point number is used for x; for example, "'plot 'file' using 1'" is identical to "'plot 'file' using 0:1'". If the Section 2.12.1.9 [using], page 61 list has two entries, these will be used for x and y. Additional entries are usually errors in x and/or y. See Section 2.21.58 [style], page 130 for details about plotting styles that make use of error information, and 'fit' for use of error information in curve fitting.

'scanf' accepts several numerical specifications but 'gnuplot' requires all inputs to be double-precision floating-point variables, so "%lf" is essentially the only permissible specifier. A format string given by the user must contain at least one such input specifier, and no more than seven of them. 'scanf' expects to see white space—a blank, tab ("\t"), newline ("\n"), or formfeed ("\f")—between numbers; anything else in the input stream must be explicitly skipped.

Note that the use of "\t", "\n", or "\f" requires use of double-quotes rather than single-quotes.

Examples:

This creates a plot of the sum of the 2nd and 3rd data against the first: The format string specifies comma- rather than space-separated columns. The same result could be achieved by specifying ‘set datafile separator ","‘.

```
plot 'file' using 1:($2+$3) '%lf,%lf,%lf'
```

In this example the data are read from the file "MyData" using a more complicated format:

```
plot 'MyData' using "%*lf%lf%*20[^\n]%lf"
```

The meaning of this format is:

```
%*lf        ignore a number
%lf         read a double-precision number (x by default)
%*20[^\n]   ignore 20 non-newline characters
%lf         read a double-precision number (y by default)
```

One trick is to use the ternary ‘?:‘ operator to filter data:

```
plot 'file' using 1:($3>10 ? $2 : 1/0)
```

which plots the datum in column two against that in column one provided the datum in column three exceeds ten. ‘1/0‘ is undefined; ‘gnuplot‘ quietly ignores undefined points, so unsuitable points are suppressed.

In fact, you can use a constant expression for the column number, provided it doesn't start with an opening parenthesis; constructs like ‘using 0+(complicated expression)‘ can be used. The crucial point is that the expression is evaluated once if it doesn't start with a left parenthesis, or once for each data point read if it does.

If timeseries data are being used, the time can span multiple columns. The starting column should be specified. Note that the spaces within the time must be included when calculating starting columns for other data. E.g., if the first element on a line is a time with an embedded space, the y value should be specified as column three.

It should be noted that ‘plot 'file'‘, ‘plot 'file' using 1:2‘, and ‘plot 'file' using ($1):($2)‘ can be subtly different: 1) if Section 2.21.50.4 [file], page 125 has some lines with one column and some with two, the first will invent x values when they are missing, the second will quietly ignore the lines with one column, and the third will store an undefined value for lines with one point (so that in a plot with lines, no line joins points across the bad point); 2) if a line contains text at the first column, the first will abort the plot on an error, but the second and third should quietly skip the garbage.

In fact, it is often possible to plot a file with lots of lines of garbage at the top simply by specifying

```
plot 'file' using 1:2
```

However, if you want to leave text in your data files, it is safer to put the comment character (#) in the first column of the text lines. Feeble using demos.

If gnuplot is built with configuration option –enable-datastrings, then additional modifiers to Section 2.12.1.9 [using], page 61 can specify handling of text fields in the datafile. See ‘datastrings‘, ‘using xticlabels‘, ‘using title‘.

— USING TITLE —

If gnuplot is built with configuration option –enable-datastrings, then the first entry of a column of the input data file can be used as a string to provide the plot title in the key box. The column containing specified is independent of the column[s] used for the plot itself.

```
plot 'data' using 1:($2/$3) title 2
```

In this case the entry in the first row of column 2 will be used for the key entry of the plot constructed from dividing column 2 by column 3. The entry in the first row of column 3 will be ignored.

— XTICLABELS —

If gnuplot is built with configuration option –enable-datastrings, then a column of the input data file can be used to label axis tic marks. The format of such a plot command is

```
plot 'datafile' us-
ing <xcol>:<ycol>:xticlabels(<labelcol>) with <plotstyle>
```

Tic labels may be read for any of the plot axes: x x2 y y2 z. The ‘ticlabels(<labelcol>)‘ specifiers must come after all of the data coordinate specifiers in the Section 2.12.1.9 [using], page 61 portion of the command. For each data point which has a valid set of X,Y[,Z] coordinates, the text field found in column <labelcol> is added to the list of xtic labels at the same X coordinate as the point it belongs to. ‘xticlabels(<labelcol>)‘ may be shortened to ‘xtic(<labelcol>)‘.

Example:

```
splot "data" using 2:4:6:xtic(1):ytic(3):ztic(6)
```

In this example the x and y axis tic labels are taken from different columns than the x and y coordinate values. The z axis tics, however, are generated from the z coordinate of the corresponding point.

— X2TICLABELS —

See ‘plot using xticlabels‘.

— YTICLABELS —

See ‘plot using xticlabels‘.

— Y2TICLABELS —

See ‘plot using xticlabels‘.

— ZTICLABELS —

See ‘plot using xticlabels‘.

2.12.2 errorbars

Error bars are supported for 2-d data file plots by reading one to four additional columns (or Section 2.12.1.9 [using], page 61 entries); these additional values are used in different ways by the various errorbar styles.

In the default situation, 'gnuplot' expects to see three, four, or six numbers on each line of the data file—either

```
(x, y, ydelta),
(x, y, ylow, yhigh),
(x, y, xdelta),
(x, y, xlow, xhigh),
(x, y, xdelta, ydelta), or
(x, y, xlow, xhigh, ylow, yhigh).
```

The x coordinate must be specified. The order of the numbers must be exactly as given above, though the Section 2.12.1.9 [using], page 61 qualifier can manipulate the order and provide values for missing columns. For example,

```
plot 'file' with errorbars
plot 'file' using 1:2:(sqrt($1)) with xerrorbars
plot 'file' using 1:2:($1-$3):($1+$3):4:5 with xyerrorbars
```

The last example is for a file containing an unsupported combination of relative x and absolute y errors. The Section 2.12.1.9 [using], page 61 entry generates absolute x min and max from the relative error.

The y error bar is a vertical line plotted from (x, ylow) to (x, yhigh). If ydelta is specified instead of ylow and yhigh, ylow = y - ydelta and yhigh = y + ydelta are derived. If there are only two numbers on the record, yhigh and ylow are both set to y. The x error bar is a horizontal line computed in the same fashion. To get lines plotted between the data points, 'plot' the data file twice, once with errorbars and once with lines (but remember to use the 'notitle' option on one to avoid two entries in the key). Alternately, use the errorlines command (see Section 2.12.3 [errorlines], page 64).

The error bars have crossbars at each end unless Section 2.21.4 [bars], page 79 is used (see Section 2.21.4 [bars], page 79 for details).

If autoscaling is on, the ranges will be adjusted to include the error bars.

See also errorbar demos.

See Section 2.12.1.9 [using], page 61, Section 2.12.7 [with], page 68, and Section 2.21.58 [style], page 130 for more information.

2.12.3 errorlines

Lines with error bars are supported for 2-d data file plots by reading one to four additional columns (or Section 2.12.1.9 [using], page 61 entries); these additional values are used in different ways by the various errorlines styles.

In the default situation, 'gnuplot' expects to see three, four, or six numbers on each line of the data file—either

```
(x, y, ydelta),
(x, y, ylow, yhigh),
(x, y, xdelta),
(x, y, xlow, xhigh),
(x, y, xdelta, ydelta), or
(x, y, xlow, xhigh, ylow, yhigh).
```

The x coordinate must be specified. The order of the numbers must be exactly as given above, though the Section 2.12.1.9 [using], page 61 qualifier can manipulate the order and provide values for missing columns. For example,

```
plot 'file' with errorlines
plot 'file' using 1:2:(sqrt($1)) with xerrorlines
plot 'file' using 1:2:($1-$3):($1+$3):4:5 with xyerrorlines
```

The last example is for a file containing an unsupported combination of relative x and absolute y errors. The Section 2.12.1.9 [using], page 61 entry generates absolute x min and max from the relative error.

The y error bar is a vertical line plotted from (x, ylow) to (x, yhigh). If ydelta is specified instead of ylow and yhigh, ylow = y - ydelta and yhigh = y + ydelta are derived. If there are only two numbers on the record, yhigh and ylow are both set to y. The x error bar is a horizontal line computed in the same fashion.

The error bars have crossbars at each end unless Section 2.21.4 [bars], page 79 is used (see Section 2.21.4 [bars], page 79 for details).

If autoscaling is on, the ranges will be adjusted to include the error bars.

See Section 2.12.1.9 [using], page 61, Section 2.12.7 [with], page 68, and Section 2.21.58 [style], page 130 for more information.

2.12.4 parametric

When in parametric mode ('set parametric') mathematical expressions must be given in pairs for 'plot' and in triplets for 'splot'.

Examples:

```
plot sin(t),t**2
splot cos(u)*cos(v),cos(u)*sin(v),sin(u)
```

Data files are plotted as before, except any preceding parametric function must be fully specified before a data file is given as a plot. In other words, the x parametric function ('sin(t)' above) and the y parametric function ('t**2' above) must not be interrupted with any modifiers or data functions; doing so will generate a syntax error stating that the parametric function is not fully specified.

Other modifiers, such as Section 2.12.7 [with], page 68 and 'title', may be specified only after the parametric function has been completed:

```
plot sin(t),t**2 title 'Parametric example' with linespoints
```

See also Parametric Mode Demos.

2.12.5 ranges

The optional ranges specify the region of the graph that will be displayed.

Syntax:

```
[{<dummy-var>=}{{<min>}:{<max>}}]
[{{<min>}:{<max>}}]
```

The first form applies to the independent variable (Section 2.21.87 [xrange], page 154 or Section 2.21.70 [trange], page 149, if in parametric mode). The second form applies to the dependent variable Section 2.21.102 [yrange], page 160 (and Section 2.21.87 [xrange], page 154, too, if in parametric mode). <dummy-var> is a new name for the independent variable. (The defaults may be changed with Section 2.21.17 [dummy], page 91.) The optional <min> and <max> terms can be constant expressions or *.

In non-parametric mode, the order in which ranges must be given is Section 2.21.87 [xrange], page 154 and Section 2.21.102 [yrange], page 160.

In parametric mode, the order for the ‘plot‘ command is Section 2.21.70 [trange], page 149, Section 2.21.87 [xrange], page 154, and Section 2.21.102 [yrange], page 160. The following ‘plot‘ command shows setting the Section 2.21.70 [trange], page 149 to [-pi:pi], the Section 2.21.87 [xrange], page 154 to [-1.3:1.3] and the Section 2.21.102 [yrange], page 160 to [-1:1] for the duration of the graph:

```
plot [-pi:pi] [-1.3:1.3] [-1:1] sin(t),t**2
```

Note that the x2range and y2range cannot be specified here—Section 2.21.80 [x2range], page 151 and Section 2.21.95 [y2range], page 159 must be used.

Ranges are interpreted in the order listed above for the appropriate mode. Once all those needed are specified, no further ones must be listed, but unneeded ones cannot be skipped—use an empty range ‘[]‘ as a placeholder.

‘*‘ can be used to allow autoscaling of either of min and max. See also Section 2.21.3 [autoscale], page 77.

Ranges specified on the ‘plot‘ or ‘splot‘ command line affect only that graph; use the Section 2.21.87 [xrange], page 154, Section 2.21.102 [yrange], page 160, etc., commands to change the default ranges for future graphs.

With time data, you must provide the range (in the same manner as the time appears in the datafile) within quotes. ‘gnuplot‘ uses the Section 2.21.67 [timefmt], page 147 string to read the value—see Section 2.21.67 [timefmt], page 147.

Examples:

This uses the current ranges:

```
plot cos(x)
```

This sets the x range only:

```
plot [-10:30] sin(pi*x)/(pi*x)
```

This is the same, but uses t as the dummy-variable:

```
plot [t = -10 :30]  sin(pi*t)/(pi*t)
```

This sets both the x and y ranges:

```
plot [-pi:pi] [-3:3]  tan(x), 1/x
```

This sets only the y range, and turns off autoscaling on both axes:

```
plot [ ] [-2:sin(5)*-8] sin(x)**besj0(x)
```

This sets xmax and ymin only:

```
plot [:200] [-pi:]  exp(sin(x))
```

This sets the x range for a timeseries:

```
set timefmt "%d/%m/%y %H:%M"
plot ["1/6/93 12:00":"5/6/93 12:00"] 'timedata.dat'
```

2.12.6 title

A line title for each function and data set appears in the key, accompanied by a sample of the line and/or symbol used to represent it. It can be changed by using the 'title' option.

Syntax:

```
title "<title>" | notitle ["<ignored title>"]
```

where <title> is the new title of the line and must be enclosed in quotes. The quotes will not be shown in the key. A special character may be given as a backslash followed by its octal value ("\345"). The tab character "\t" is understood. Note that backslash processing occurs only for strings enclosed in double quotes—use single quotes to prevent such processing. The newline character "\n" is not processed in key entries in either type of string.

The line title and sample can be omitted from the key by using the keyword 'notitle'. A null title ('title "') is equivalent to 'notitle'. If only the sample is wanted, use one or more blanks ('title ' ''). If 'notitle' is followed by a string this string is ignored.

If 'key autotitles' is set (which is the default) and neither 'title' nor 'notitle' are specified the line title is the function name or the file name as it appears on the 'plot' command. If it is a file name, any datafile modifiers specified will be included in the default title.

The layout of the key itself (position, title justification, etc.) can be controlled by Section 2.21.28 [key], page 100. Please see Section 2.21.28 [key], page 100 for details.

Examples:

This plots y=x with the title 'x':

```
plot x
```

This plots x squared with title "x^2" and file "data.1" with title "measured data":

```
plot x**2 title "x^2", 'data.1' t "measured data"
```

This puts an untitled circular border around a polar graph:

```
set polar; plot my_function(t), 1 notitle
```

2.12.7 with

Functions and data may be displayed in one of a large number of styles. The Section 2.12.7 [with], page 68 keyword provides the means of selection.

Syntax:

```
with <style> { {linestyle | ls <line_style>}
               | {{linetype  | lt <line_type>}
                  {linewidth | lw <line_width>}
                  {linecolor | lc <colorspec>}
                  {pointtype | pt <point_type>}
                  {pointsize | ps <point_size>}
                  {fill | fs <fillstyle>}
                  {nohidden3d}
                  {palette}}
             }
```

where <style> is either 'lines', 'points', 'linespoints', 'impulses', 'dots', 'steps', 'fsteps', 'histeps', Section 2.12.2 [errorbars], page 64, 'labels', 'xerrorbars', 'yerrorbars', 'xyerrorbars', Section 2.12.3 [errorlines], page 64, 'xerrorlines', 'yerrorlines', 'xyerrorlines', 'boxes', 'histograms', 'filledcurves', 'boxerrorbars', 'boxxyerrorbars', 'financebars', 'candlesticks', 'vectors', 'image', 'rgbimage' or pm3d. Some of these styles require additional information. See 'plotting styles' for details of each style. 'fill' is relevant only to certain 2D plots (currently 'boxes' 'boxxyerrorbars' and 'candlesticks'). Note that 'filledcurves' and pm3d can take an additional option not listed above (the latter only when used in the 'splot' command)—see their help or examples below for more details.

Default styles are chosen with the 'set style function' and 'set style data' commands.

By default, each function and data file will use a different line type and point type, up to the maximum number of available types. All terminal drivers support at least six different point types, and re-use them, in order, if more are required. The LaTeX driver supplies an additional six point types (all variants of a circle), and thus will only repeat after 12 curves are plotted with points. The PostScript drivers (Section 2.21.50.6 [postscript], page 126) supplies a total of 64.

If you wish to choose the line or point type for a single plot, <line_type> and <point_type> may be specified. These are positive integer constants (or expressions) that specify the line type and point type to be used for the plot. Use Section 2.25 [test], page 168 to display the types available for your terminal.

You may also scale the line width and point size for a plot by using <line_width> and <point_size>, which are specified relative to the default values for each terminal. The pointsize may also be altered globally—see Section 2.21.51 [pointsize], page 126 for details. But note that both <point_size> as set here and as set by Section 2.21.51 [pointsize], page 126 multiply the default point size—their effects are not cumulative. That is, 'set pointsize 2; plot x w p ps 3' will use points three times default size, not six.

It is also possible to specify 'pointsize variable' either as part of a line style or for an individual plot. In this case one extra column of input is required, i.e. 3 columns for a 2D plot and 4 columns for a 3D splot. The size of each individual point is determined by multiplying the global pointsize by the value read from the data file.

If you have defined specific line type/width and point type/size combinations with 'set style line', one of these may be selected by setting <line_style> to the index of the desired style.

If gnuplot was built with pm3d support, the special keyword Section 2.21.50 [palette], page 120 is allowed for smooth color change of lines, points and dots in 'splots'. The color is chosen from a smooth palette which was set previously with the command Section 2.21.50 [palette], page 120. The color value corresponds to the z-value of the point coordinates or to the color coordinate if specified by the 4th parameter in Section 2.12.1.9 [using], page 61. Both 2d and 3d plots ('plot' and 'splot' commands) can use palette colors as specified by either their fractional value or the corresponding value mapped to the colorbox range. 2d plots can not use palette colors mapped by Z value. See 'colors', Section 2.21.50 [palette], page 120, 'linetype'.

The keyword 'nohidden3d' applies only to plots made with the 'splot' command. Normally the global option Section 2.21.25 [hidden3d], page 97 applies to all plots in the graph. You can attach the 'nohidden3d' option to any individual plots that you want to exclude from the hidden3d processing. The individual elements (lines, dots, labels, ...) of a plot marked 'nohidden3d' will all be drawn, even if they would normally be obscured by the elements of some other plot.

The keywords may be abbreviated as indicated.

Note that the 'linewidth', Section 2.21.51 [pointsize], page 126 and Section 2.21.50 [palette], page 120 options are not supported by all terminals.

Examples:

This plots sin(x) with impulses:

```
plot sin(x) with impulses
```

This plots x with points, x**2 with the default:

```
plot x w points, x**2
```

This plots tan(x) with the default function style, file "data.1" with lines:

```
plot [ ] [-2:5] tan(x), 'data.1' with l
```

This plots "leastsq.dat" with impulses:

```
plot 'leastsq.dat' w i
```

This plots the data file "population" with boxes:

```
plot 'population' with boxes
```

This plots "exper.dat" with errorbars and lines connecting the points (errorbars require three or four columns):

```
plot 'exper.dat' w lines, 'exper.dat' notitle w errorbars
```

Another way to plot "exper.dat" with errorlines (errorbars require three or four columns):

```
plot 'exper.dat' w errorlines
```

This plots sin(x) and cos(x) with linespoints, using the same line type but different point types:

```
plot sin(x) with linesp lt 1 pt 3, cos(x) with linesp lt 1 pt 4
```

This plots file "data" with points of type 3 and twice usual size:

```
plot 'data' with points pointtype 3 pointsize 2
```

This plots file "data" with variable pointsize read from column 4

```
plot 'data' using 1:2:4 with points pt 5 pointsize variable
```

This plots two data sets with lines differing only by weight:

```
plot 'd1' t "good" w l lt 2 lw 3, 'd2' t "bad" w l lt 2 lw 1
```

This plots filled curve of x*x and a color stripe:

```
plot x*x with filledcurve closed, 40 with filledcurve y1=10
```

This plots x*x and a color box:

```
plot x*x, (x>=-5 && x<=5 ? 40 : 1/0) with filledcurve y1=10 lt 8
```

This plots a surface with color lines:

```
splot x*x-y*y with line palette
```

This plots two color surfaces at different altitudes:

```
splot x*x-y*y with pm3d, x*x+y*y with pm3d at t
```

2.13 print

The 'print' command prints the value of <expression> to the screen. It is synonymous with 'pause 0'. <expression> may be anything that 'gnuplot' can evaluate that produces a number, or it can be a string.

Syntax:

```
print <expression> {, <expression>, ...}
```

See 'expressions'. The output file can be set with 'set print'.

2.14 pwd

The Section 2.14 [pwd], page 71 command prints the name of the working directory to the screen.

2.15 quit

The Section 2.4 [exit], page 36 and Section 2.15 [quit], page 71 commands and END-OF-FILE character will exit ‘gnuplot‘. Each of these commands will clear the output device (as does the Section 2.3 [clear], page 36 command) before exiting.

2.16 raise

Syntax:

```
raise {x11_plot_window_nb}
```

The Section 2.16 [raise], page 71 command raises (opposite to Section 2.10 [lower], page 47) plot window(s) associated with the interactive terminal of your gnuplot session, i.e. ‘pm‘, ‘win‘ or ‘x11‘. It puts the plot window to front (top) in the z-order windows stack of the window manager of your desktop.

As ‘x11‘ supports multiple plot windows, then by default it raises these windows in descending order of most recently created on top to the least recently created on bottom. If a plot number is supplied as an optional parameter, only the associated plot window will be raised if it exists.

The optional parameter is ignored for single plot-windows terminal, i.e. ‘pm‘ and ‘win‘.

2.17 replot

The Section 2.17 [replot], page 71 command without arguments repeats the last ‘plot‘ or ‘splot‘ command. This can be useful for viewing a plot with different ‘set‘ options, or when generating the same plot for several devices.

Arguments specified after a Section 2.17 [replot], page 71 command will be added onto the last ‘plot‘ or ‘splot‘ command (with an implied ’,’ separator) before it is repeated. Section 2.17 [replot], page 71 accepts the same arguments as the ‘plot‘ and ‘splot‘ commands except that ranges cannot be specified. Thus you can use Section 2.17 [replot], page 71 to plot a function against the second axes if the previous command was ‘plot‘ but not if it was ‘splot‘.

N.B.—use of

```
plot '-' ; ... ; replot
```

is not recommended. ‘gnuplot‘ does not store the inline data internally, so since Section 2.17 [replot], page 71 appends new information to the previous ‘plot‘ and then executes the modified command, the ‘’-’‘ from the initial ‘plot‘ will expect to read inline data again.

Note that Section 2.17 [replot], page 71 does not work in Section 2.21.38 [multiplot], page 111 mode, since it reproduces only the last plot rather than the entire screen.

See also ‘command-line-editing‘ for ways to edit the last ‘plot‘ (‘splot‘) command.

See also ‘show plot‘ to show the whole current plotting command, and the possibility to copy it into the ‘history‘.

2.18 reread

The Section 2.18 [reread], page 72 command causes the current ‘gnuplot‘ command file, as specified by a ‘load‘ command or on the command line, to be reset to its starting point before further commands are read from it. This essentially implements an endless loop of the commands from the beginning of the command file to the Section 2.18 [reread], page 72 command. (But this is not necessarily a disaster—Section 2.18 [reread], page 72 can be very useful when used in conjunction with Section 2.8 [if], page 46. See Section 2.8 [if], page 46 for details.) The Section 2.18 [reread], page 72 command has no effect if input from standard input.

Examples:

Suppose the file "looper" contains the commands

```
a=a+1
plot sin(x*a)
pause -1
if(a<5) reread
```

and from within ‘gnuplot‘ you submit the commands

```
a=0
load 'looper'
```

The result will be four plots (separated by the Section 2.11 [pause], page 47 message).

Suppose the file "data" contains six columns of numbers with a total yrange from 0 to 10; the first is x and the next are five different functions of x. Suppose also that the file "plotter" contains the commands

```
c_p = c_p+1
plot "$0" using 1:c_p with lines linetype c_p
if(c_p <  n_p) reread
```

and from within ‘gnuplot‘ you submit the commands

```
n_p=6
c_p=1
unset key
set yrange [0:10]
set multiplot
call 'plotter' 'data'
unset multiplot
```

The result is a single graph consisting of five plots. The yrange must be set explicitly to guarantee that the five separate graphs (drawn on top of each other in multiplot mode) will have exactly the same axes. The linetype must be specified; otherwise all the plots would be drawn with the same type. See animate.dem in demo directory for an animated example.

2.19 reset

The Section 2.19 [reset], page 73 command causes all graph-related options that can be set with the ‘set‘ command to take on their default values. This command is useful, e.g., to restore the default graph settings at the end of a command file, or to return to a defined state after lots of settings have been changed within a command file. Please refer to the ‘set‘ command to see the default values that the various options take.

The following ‘set‘ commands do not change the graph status and are thus left unchanged: the terminal set with ‘set term‘, the output file set with Section 2.21.46 [output], page 115 and directory paths set with Section 2.21.31 [loadpath], page 106 and Section 2.21.20 [fontpath], page 93.

2.20 save

The Section 2.20 [save], page 73 command saves user-defined functions, variables, the ‘set term‘ status, all ‘set‘ options, or all of these, plus the last ‘plot‘ (‘splot‘) command to the specified file.

Syntax:

```
save  {<option>} '<filename>'
```

where <option> is ‘functions‘, Section 2.21.72 [variables], page 149, ‘terminal‘ or ‘set‘. If no option is used, ‘gnuplot‘ saves functions, variables, ‘set‘ options and the last ‘plot‘ (‘splot‘) command.

Section 2.20 [save], page 73d files are written in text format and may be read by the ‘load‘ command. For Section 2.20 [save], page 73 with the ‘set‘ option or without any option, the ‘terminal‘ choice and the Section 2.21.46 [output], page 115 filename are written out as a comment, to get an output file that works in other installations of gnuplot, without changes and without risk of unwillingly overwriting files.

‘save terminal‘ will write out just the ‘terminal‘ status, without the comment marker in front of it. This is mainly useful for switching the ‘terminal‘ setting for a short while, and getting back to the previously set terminal, afterwards, by loading the saved ‘terminal‘ status. Note that for a single gnuplot session you may rather use the other method of saving and restoring current terminal by the commands ‘set term push‘ and ‘set term pop‘, see ‘set term‘.

The filename must be enclosed in quotes.

The special filename `"-"` may be used to Section 2.20 [save], page 73 commands to standard output. On systems which support a popen function (Unix), the output of save can be piped through an external program by starting the file name with a `'|'`. This provides a consistent interface to ‘gnuplot‘’s internal settings to programs which communicate with ‘gnuplot‘ through a pipe. Please see help for ‘batch/interactive‘ for more details.

Examples:

```
save 'work.gnu'
save functions 'func.dat'
save var 'var.dat'
save set 'options.dat'
save term 'myterm.gnu'
```

```
save '-'
save '|grep title >t.gp'
```

2.21 set-show

The ‘set‘ command can be used to set _lots_ of options. No screen is drawn, however, until a ‘plot‘, ‘splot‘, or Section 2.17 [replot], page 71 command is given.

The ‘show‘ command shows their settings; ‘show all‘ shows all the settings.

Options changed using ‘set‘ can be returned to the default state by giving the corresponding Section 2.26 [unset], page 169 command. See also the Section 2.19 [reset], page 73 command, which returns all settable parameters to default values.

If a variable contains time/date data, ‘show‘ will display it according to the format currently defined by Section 2.21.67 [timefmt], page 147, even if that was not in effect when the variable was initially defined.

2.21.1 angles

By default, ‘gnuplot‘ assumes the independent variable in polar graphs is in units of radians. If ‘set angles degrees‘ is specified before ‘set polar‘, then the default range is [0:360] and the independent variable has units of degrees. This is particularly useful for plots of data files. The angle setting also applies to 3-d mapping as set via the Section 2.21.35 [mapping], page 108 command.

Syntax:

```
set angles {degrees | radians}
show angles
```

The angle specified in ‘set grid polar‘ is also read and displayed in the units specified by Section 2.21.1 [angles], page 74.

Section 2.21.1 [angles], page 74 also affects the arguments of the machine-defined functions sin(x), cos(x) and tan(x), and the outputs of asin(x), acos(x), atan(x), atan2(x), and arg(x). It has no effect on the arguments of hyperbolic functions or Bessel functions. However, the output arguments of inverse hyperbolic functions of complex arguments are affected; if these functions are used, ‘set angles radians‘ must be in effect to maintain consistency between input and output arguments.

```
x={1.0,0.1}
set angles radians
y=sinh(x)
print y         #prints {1.16933, 0.154051}
print asinh(y)  #prints {1.0, 0.1}
```

but

```
set angles degrees
y=sinh(x)
print y         #prints {1.16933, 0.154051}
print asinh(y)  #prints {57.29578, 5.729578}
```

See also poldat.dem: polar plot using Section 2.21.1 [angles], page 74 demo.

2.21.2 arrow

Arbitrary arrows can be placed on a plot using the Section 2.21.2 [arrow], page 75 command.

Syntax:

```
set arrow {<tag>} {from <position>} {to|rto <position>}
          { {arrowstyle | as <arrow_style>}
            | { {nohead | head | heads}
                {size <length>,<angle>{,<backangle>}}
                {filled | empty | nofilled}
                {front | back}
                { {linestyle | ls <line_style>}
                  | {linetype | lt <line_type>}
                    {linewidth | lw <line_width} } } }


unset arrow {<tag>}
show arrow {<tag>}
```

<tag> is an integer that identifies the arrow. If no tag is given, the lowest unused tag value is assigned automatically. The tag can be used to delete or change a specific arrow. To change any attribute of an existing arrow, use the Section 2.21.2 [arrow], page 75 command with the appropriate tag and specify the parts of the arrow to be changed.

The <position>s are specified by either x,y or x,y,z, and may be preceded by ‘first‘, ‘second‘, ‘graph‘, ‘screen‘, or ‘character‘ to select the coordinate system. Unspecified coordinates default to 0. The end points can be specified in one of five coordinate systems—‘first‘ or ‘second‘ axes, ‘graph‘, ‘screen‘, or ‘character‘. See ‘coordinates‘ for details. A coordinate system specifier does not carry over from the "from" position to the "to" position. Arrows outside the screen boundaries are permitted but may cause device errors. If the end point is specified by "rto" instead of "to" it is drawn relatively to the start point. For linear axes, ‘graph‘ and ‘screen‘ coordinates, the distance between the start and the end point corresponds to the given relative coordinate. For logarithmic axes, the relative given coordinate corresponds to the factor of the coordinate between start and end point. Thus, a negative relative value or zero are not allowed for logarithmic axes.

Specifying ‘nohead‘ produces an arrow drawn without a head—a line segment. This gives you yet another way to draw a line segment on the plot. By default, arrows have heads. Specifying ‘heads‘ draws arrow heads on both ends of the line. Not all terminal types support double-ended arrows.

Head size can be controlled by ‘size <length>,<angle>‘ or ‘size <length>,<angle>,<backangle>‘, where ‘<length>‘ defines length of each branch of the arrow head and ‘<angle>‘ the angle (in degrees) they make with the arrow. ‘<Length>‘ is in x-axis units; this can be changed by ‘first‘, ‘second‘, ‘graph‘, ‘screen‘, or ‘character‘ before the <length>; see ‘coordinates‘ for details. ‘<Backangle>‘ only takes effect when ‘filled‘ or ‘empty‘ is also used. Then, ‘<backangle>‘ is the angle (in degrees) the back branches make with the arrow (in the same direction as ‘<angle>‘). The ‘fig‘ terminal has a restricted backangle function. It supports three different angles. There are two thresholds: Below 70 degrees, the arrow head gets an indented back angle. Above 110 degrees, the arrow head has an acute back angle. Between these thresholds, the back line is straight.

Specifying 'filled' produces filled arrow heads (if heads are used). Filling is supported on filled-polygon capable terminals, see help of pm3d for their list, otherwise the arrow heads are closed but not filled. The same result (closed but not filled arrow head) is reached by specifying 'empty'. Further, filling and outline is obviously not supported on terminals drawing arrows by their own specific routines, like 'metafont', 'metapost', 'latex' or 'tgif'.

The line style may be selected from a user-defined list of line styles (see 'set style line') or may be defined here by providing values for <line_type> (an index from the default list of styles) and/or <line_width> (which is a multiplier for the default width).

Note, however, that if a user-defined line style has been selected, its properties (type and width) cannot be altered merely by issuing another Section 2.21.2 [arrow], page 75 command with the appropriate index and 'lt' or 'lw'.

If 'front' is given, the arrow is written on top of the graphed data. If 'back' is given (the default), the arrow is written underneath the graphed data. Using 'front' will prevent an arrow from being obscured by dense data.

Examples:

To set an arrow pointing from the origin to (1,2) with user-defined style 5, use:

```
set arrow to 1,2 ls 5
```

To set an arrow from bottom left of plotting area to (-5,5,3), and tag the arrow number 3, use:

```
set arrow 3 from graph 0,0 to -5,5,3
```

To change the preceding arrow to end at 1,1,1, without an arrow head and double its width, use:

```
set arrow 3 to 1,1,1 nohead lw 2
```

To draw a vertical line from the bottom to the top of the graph at x=3, use:

```
set arrow from 3, graph 0 to 3, graph 1 nohead
```

To draw a vertical arrow with T-shape ends, use:

```
set arrow 3 from 0,-5 to 0,5 heads size screen 0.1,90
```

To draw an arrow relatively to the start point, where the relative distances are given in graph coordinates, use:

```
set arrow from 0,-5 rto graph 0.1,0.1
```

To draw an arrow with relative end point in logarithmic x axis, use:

```
set logscale x
set arrow from 100,-5 rto 10,10
```

This draws an arrow from 100,-5 to 1000,5. For the logarithmic x axis, the relative coordinate 10 means "factor 10" while for the linear y axis, the relative coordinate 10 means "difference 10".

To delete arrow number 2, use:

```
unset arrow 2
```

To delete all arrows, use:

```
unset arrow
```

To show all arrows (in tag order), use:

```
show arrow
```

arrows demos.

2.21.3 autoscale

Autoscaling may be set individually on the x, y or z axis or globally on all axes. The default is to autoscale all axes.

Syntax:

```
      set au-
toscale {<axes>{|min|max|fixmin|fixmax|fix} | fix | keepfix}
      unset autoscale {<axes>}
      show autoscale
```

where <axes> is either ‘x‘, ‘y‘, ‘z‘, ‘cb‘, ‘x2‘, ‘y2‘ or ‘xy‘. A keyword with ‘min‘ or ‘max‘ appended (this cannot be done with ‘xy‘) tells ‘gnuplot‘ to autoscale just the minimum or maximum of that axis. If no keyword is given, all axes are autoscaled.

A keyword with ‘fixmin‘, ‘fixmax‘ or ‘fix‘ appended tells gnuplot to disable extension of the axis range to the next tic mark position, for autoscaled axes using equidistant tics; ‘set autoscale fix‘ sets this for all axes. Command ‘set autoscale keepfix‘ autoscales all axes while keeping the fix settings.

When autoscaling, the axis range is automatically computed and the dependent axis (y for a ‘plot‘ and z for ‘splot‘) is scaled to include the range of the function or data being plotted.

If autoscaling of the dependent axis (y or z) is not set, the current y or z range is used.

Autoscaling the independent variables (x for ‘plot‘ and x,y for ‘splot‘) is a request to set the domain to match any data file being plotted. If there are no data files, autoscaling an independent variable has no effect. In other words, in the absence of a data file, functions alone do not affect the x range (or the y range if plotting z = f(x,y)).

Please see Section 2.21.87 [xrange], page 154 for additional information about ranges.

The behavior of autoscaling remains consistent in parametric mode, (see ‘set parametric‘). However, there are more dependent variables and hence more control over x, y, and z axis scales. In parametric mode, the independent or dummy variable is t for ‘plot‘s and u,v for ‘splot‘s. Section 2.21.3 [autoscale], page 77 in parametric mode, then, controls all ranges (t, u, v, x, y, and z) and allows x, y, and z to be fully autoscaled.

Autoscaling works the same way for polar mode as it does for parametric mode for ‘plot‘, with the extension that in polar mode Section 2.21.17 [dummy], page 91 can be used to change the independent variable from t (see Section 2.21.17 [dummy], page 91).

When tics are displayed on second axes but no plot has been specified for those axes, x2range and y2range are inherited from xrange and yrange. This is done _before_ xrange and yrange are autoextended to a whole number of tics, which can cause unexpected results. You can use the 'fixmin' or 'fixmax' options to avoid this.

Examples:

This sets autoscaling of the y axis (other axes are not affected):

```
set autoscale y
```

This sets autoscaling only for the minimum of the y axis (the maximum of the y axis and the other axes are not affected):

```
set autoscale ymin
```

This disables extension of the x2 axis tics to the next tic mark, thus keeping the exact range as found in the plotted data and functions:

```
set autoscale x2fixmin
set autoscale x2fixmax
```

This sets autoscaling of the x and y axes:

```
set autoscale xy
```

This sets autoscaling of the x, y, z, x2 and y2 axes:

```
set autoscale
```

This disables autoscaling of the x, y, z, x2 and y2 axes:

```
unset autoscale
```

This disables autoscaling of the z axis only:

```
unset autoscale z
```

2.21.3.1 parametric mode

When in parametric mode ('set parametric'), the xrange is as fully scalable as the y range. In other words, in parametric mode the x axis can be automatically scaled to fit the range of the parametric function that is being plotted. Of course, the y axis can also be automatically scaled just as in the non-parametric case. If autoscaling on the x axis is not set, the current x range is used.

Data files are plotted the same in parametric and non-parametric mode. However, there is a difference in mixed function and data plots: in non-parametric mode with autoscaled x, the x range of the datafile controls the x range of the functions; in parametric mode it has no influence.

For completeness a last command 'set autoscale t' is accepted. However, the effect of this "scaling" is very minor. When 'gnuplot' determines that the t range would be empty, it makes a small adjustment if autoscaling is true. Otherwise, 'gnuplot' gives an error. Such behavior may, in fact, not be very useful and the command 'set autoscale t' is certainly questionable.

'splot' extends the above ideas as you would expect. If autoscaling is set, then x, y, and z ranges are computed and each axis scaled to fit the resulting data.

2.21.3.2 polar mode

When in polar mode ('set polar'), the xrange and the yrange are both found from the polar coordinates, and thus they can both be automatically scaled. In other words, in polar mode both the x and y axes can be automatically scaled to fit the ranges of the polar function that is being plotted.

When plotting functions in polar mode, the rrange may be autoscaled. When plotting data files in polar mode, the trange may also be autoscaled. Note that if the trange is contained within one quadrant, autoscaling will produce a polar plot of only that single quadrant.

Explicitly setting one or two ranges but not others may lead to unexpected results. See also polar demos.

2.21.4 bars

The Section 2.21.4 [bars], page 79 command controls the tics at the ends of error bars, and also the width of the boxes in plot styles candlesticks and financebars.

Syntax:

```
set bars {small | large | fullwidth | <size>}
unset bars
show bars
```

'small' is a synonym for 0.0, and 'large' for 1.0. The default is 1.0 if no size is given.

The keyword 'fullwidth' is relevant only to histograms with errorbars. It sets the width of the errorbar ends to be the same as the width of the associated box in the histogram. It does not change the width of the box itself.

2.21.5 bmargin

The command Section 2.21.5 [bmargin], page 79 sets the size of the bottom margin. Please see Section 2.21.36 [margin], page 109 for details.

2.21.6 border

The Section 2.21.6 [border], page 79 and Section 2.21.6 [border], page 79 commands control the display of the graph borders for the 'plot' and 'splot' commands. Note that the borders do not necessarily coincide with the axes; with 'plot' they often do, but with 'splot' they usually do not.

Syntax:

```
      set border {<inte-
ger>} {front | back} {linewidth | lw <line_width>}
                 {{linestyle | ls <line_style>} | {line-
type | lt <line_type>}}
```

```
unset border
show border
```

With a ‘splot‘ displayed in an arbitrary orientation, like ‘set view 56,103‘, the four corners of the x-y plane can be referred to as "front", "back", "left" and "right". A similar set of four corners exist for the top surface, of course. Thus the border connecting, say, the back and right corners of the x-y plane is the "bottom right back" border, and the border connecting the top and bottom front corners is the "front vertical". (This nomenclature is defined solely to allow the reader to figure out the table that follows.)

The borders are encoded in a 12-bit integer: the bottom four bits control the border for ‘plot‘ and the sides of the base for ‘splot‘; the next four bits control the verticals in ‘splot‘; the top four bits control the edges on top of the ‘splot‘. In detail, ‘<integer>‘ should be the sum of the appropriate entries from the following table:

Bit	plot	splot
1	bottom	bottom left front
2	left	bottom left back
4	top	bottom right front
8	right	bottom right back
16	no effect	left vertical
32	no effect	back vertical
64	no effect	right vertical
128	no effect	front vertical
256	no effect	top left back
512	no effect	top right back
1024	no effect	top left front
2048	no effect	top right front

Various bits or combinations of bits may be added together in the command.

The default is 31, which is all four sides for ‘plot‘, and base and z axis for ‘splot‘.

In 2D plots the border is normally drawn on top of all plots elements (‘front‘). If you want the border to be drawn behind the plot elements, use ‘set border back‘.

Using the optional <line_style>, <line_type> and <line_width> specifiers, the way the border lines are drawn can be influenced (limited by what the current terminal driver supports).

For ‘plot‘, tics may be drawn on edges other than bottom and left by enabling the second axes – see Section 2.21.88 [xtics], page 155 for details.

If a ‘splot‘ draws only on the base, as is the case with "‘unset surface; set contour base‘", then the verticals and the top are not drawn even if they are specified.

The ‘set grid‘ options ’back’, ’front’ and ’layerdefault’ also control the order in which the border lines are drawn with respect to the output of the plotted data.

Examples:

Draw default borders:

```
set border
```

Draw only the left and bottom (‘plot‘) or both front and back bottom left (‘splot‘) borders:

```
set border 3
```

Draw a complete box around a ‘splot‘:

```
set border 4095
```

Draw a topless box around a ‘splot‘, omitting the front vertical:

```
set border 127+256+512 # or set border 1023-128
```

Draw only the top and right borders for a ‘plot‘ and label them as axes:

```
unset xtics; unset ytics; set x2tics; set y2tics; set border 12
```

2.21.7 boxwidth

The Section 2.21.7 [boxwidth], page 81 command is used to set the default width of boxes in the ‘boxes‘, ‘boxerrorbars‘, ‘candlesticks‘ and ‘histograms‘ styles.

Syntax:

```
set boxwidth {<width>} {absolute|relative}
show boxwidth
```

By default, adjacent boxes are extended in width until they touch each other. A different default width may be specified using the Section 2.21.7 [boxwidth], page 81 command. ‘Relative‘ widths are interpreted as being a fraction of this default width.

An explicit value for the boxwidth is interpreted as being a number of units along the current x axis (‘absolute‘) unless the modifier ‘relative‘ is given. If the x axis is a log-scale (see ‘set log‘) then the value of boxwidth is truly "absolute" only at x=1; this physical width is maintained everywhere along the axis (i.e. the boxes do not become narrower the value of x increases). If the range spanned by a log scale x axis is far from x=1, some experimentation may be required to find a useful value of boxwidth.

The default is superseded by explicit width information taken from an extra data column in styles ‘boxes‘ or ‘boxerrorbars‘. In a four-column data set, the fourth column will be interpreted as the box width unless the width is set to -2.0, in which case the width will be calculated automatically. See ‘style boxes‘ and ‘style boxerrorbars‘ for more details.

To set the box width to automatic use the command

```
set boxwidth
```

or, for four-column data,

```
set boxwidth -2
```

The same effect can be achieved with the Section 2.12.1.9 [using], page 61 keyword in ‘plot‘:

```
plot 'file' using 1:2:3:4:(-2)
```

To set the box width to half of the automatic size use

```
set boxwidth 0.5 relative
```

To set the box width to an absolute value of 2 use

```
set boxwidth 2 absolute
```

2.21.8 clabel

'gnuplot' will vary the linetype used for each contour level when clabel is set. When this option on (the default), a legend labels each linestyle with the z level it represents. It is not possible at present to separate the contour labels from the surface key.

Syntax:

```
set clabel {'<format>'}
unset clabel
show clabel
```

The default for the format string is %8.3g, which gives three decimal places. This may produce poor label alignment if the key is altered from its default configuration.

The first contour linetype, or only contour linetype when clabel is off, is the surface linetype +1; contour points are the same style as surface points.

See also Section 2.21.12 [contour], page 86.

2.21.9 clip

'gnuplot' can clip data points and lines that are near the boundaries of a graph.

Syntax:

```
set clip <clip-type>
unset clip <clip-type>
show clip
```

Three clip types for points and lines are supported by 'gnuplot': 'points', 'one', and 'two'. One, two, or all three clip types may be active for a single graph. Note that clipping of color filled quadrangles drawn by pm3d maps and surfaces is not controlled by this command, but by 'set pm3d clip1in' and 'set pm3d clip4in'.

The 'points' clip type forces 'gnuplot' to clip (actually, not plot at all) data points that fall within but too close to the boundaries. This is done so that large symbols used for points will not extend outside the boundary lines. Without clipping points near the boundaries, the plot may look bad. Adjusting the x and y ranges may give similar results.

Setting the 'one' clip type causes 'gnuplot' to draw a line segment which has only one of its two endpoints within the graph. Only the in-range portion of the line is drawn. The alternative is to not draw any portion of the line segment.

Some lines may have both endpoints out of range, but pass through the graph. Setting the 'two' clip-type allows the visible portion of these lines to be drawn.

In no case is a line drawn outside the graph.

The defaults are ‘noclip points‘, ‘clip one‘, and ‘noclip two‘.

To check the state of all forms of clipping, use

```
show clip
```

For backward compatibility with older versions, the following forms are also permitted:

```
set clip
unset clip
```

Section 2.21.9 [clip], page 82 is synonymous with ‘set clip points‘; Section 2.21.9 [clip], page 82 turns off all three types of clipping.

2.21.10 cntrparam

Section 2.21.10 [cntrparam], page 83 controls the generation of contours and their smoothness for a contour plot. Section 2.21.12 [contour], page 86 displays current settings of Section 2.21.10 [cntrparam], page 83 as well as Section 2.21.12 [contour], page 86.

Syntax:

```
set cntrparam { { linear
                | cubicspline
                | bspline
                | points <n>
                | order <n>
                | levels { auto {<n>} | <n>
                           | discrete <z1> {,<z2>{,<z3>...}}
                           | incremen-
tal <start>, <incr> {,<end>}
                         }
                }
              }
show contour
```

This command has two functions. First, it sets the values of z for which contour points are to be determined (by linear interpolation between data points or function isosamples.) Second, it controls the way contours are drawn between the points determined to be of equal z. `<n>` should be an integral constant expression and <z1>, <z2> ... any constant expressions. The parameters are:

‘linear‘, ‘cubicspline‘, ‘bspline‘—Controls type of approximation or interpolation. If ‘linear‘, then straight line segments connect points of equal z magnitude. If ‘cubicspline‘, then piecewise-linear contours are interpolated between the same equal z points to form somewhat smoother contours, but which may undulate. If ‘bspline‘, a guaranteed-smoother curve is drawn, which only approximates the position of the points of equal-z.

‘points‘—Eventually all drawings are done with piecewise-linear strokes. This number controls the number of line segments used to approximate the ‘bspline‘ or ‘cubicspline‘ curve. Number of cubicspline or bspline segments (strokes) = ‘points‘ * number of linear segments.

'order'—Order of the bspline approximation to be used. The bigger this order is, the smoother the resulting contour. (Of course, higher order bspline curves will move further away from the original piecewise linear data.) This option is relevant for 'bspline' mode only. Allowed values are integers in the range from 2 (linear) to 10.

'levels'— Selection of contour levels, controlled by 'auto' (default), 'discrete', 'incremental', and <n>, number of contour levels.

For 'auto', <n> specifies a nominal number of levels; the actual number will be adjusted to give simple labels. If the surface is bounded by zmin and zmax, contours will be generated at integer multiples of dz between zmin and zmax, where dz is 1, 2, or 5 times some power of ten (like the step between two tic marks).

For 'levels discrete', contours will be generated at z = <z1>, <z2> ... as specified; the number of discrete levels sets the number of contour levels. In 'discrete' mode, any 'set cntrparam levels <n>' are ignored.

For 'incremental', contours are generated at values of z beginning at <start> and increasing by <increment>, until the number of contours is reached. <end> is used to determine the number of contour levels, which will be changed by any subsequent 'set cntrparam levels <n>'. If the z axis is logarithmic, <increment> will be interpreted as a factor, just like in Section 2.21.115 [ztics], page 162.

If the command Section 2.21.10 [cntrparam], page 83 is given without any arguments specified, the defaults are used: linear, 5 points, order 4, 5 auto levels.

Examples:

```
set cntrparam bspline
set cntrparam points 7
set cntrparam order 10
```

To select levels automatically, 5 if the level increment criteria are met:

```
set cntrparam levels auto 5
```

To specify discrete levels at .1, .37, and .9:

```
set cntrparam levels discrete .1,1/exp(1),.9
```

To specify levels from 0 to 4 with increment 1:

```
set cntrparam levels incremental  0,1,4
```

To set the number of levels to 10 (changing an incremental end or possibly the number of auto levels):

```
set cntrparam levels 10
```

To set the start and increment while retaining the number of levels:

```
set cntrparam levels incremental 100,50
```

See also Section 2.21.12 [contour], page 86 for control of where the contours are drawn, and Section 2.21.8 [clabel], page 82 for control of the format of the contour labels and linetypes.

See also contours demo (contours.dem) and contours with user defined levels demo (discrete.dem).

2.21.11 color box

The color scheme, i.e. the gradient of the smooth color with min_z and max_z values of pm3d's Section 2.21.50 [palette], page 120, is drawn in a color box unless 'unset colorbox'.

```
set colorbox
set colorbox {
           { vertical | horizontal }
           { default | user }
           { origin x, y }
           { size x, y }
           { noborder | bdefault | border [line style] }
         }
show colorbox
unset colorbox
```

Colorbox position can be 'default' or 'user'. If the latter is specified the values as given with the Section 2.21.45 [origin], page 115 and Section 2.21.57 [size], page 129 subcommands are used.

'vertical' and 'horizontal' switches the orientation of the color gradient.

'origin x, y' and 'size x, y' are used only in combination with the 'user' option. The x and y values are interpreted as screen coordinates by default, and this is the only legal option for 3D plots. 2D plots, including splot with 'set view map', allow any coordinate system to be specified. Try for example:

```
set colorbox horiz user origin .1,.02 size .8,.04
```

which will draw a horizontal gradient somewhere at the bottom of the graph.

Section 2.21.6 [border], page 79 turns the border on (this is the default). 'noborder' turns the border off. If an positive integer argument is given after Section 2.21.6 [border], page 79, it is used as a line style tag which is used for drawing the border, e.g.:

```
set style line 2604 linetype -1 linewidth .4
set colorbox border 2604
```

will use line style '2604', a thin line with the default border color (-1) for drawing the border. 'bdefault' (which is the default) will use the default border line style for drawing the border of the color box.

The axis of the color box is called 'cb' and it is controlled by means of the usual axes commands, i.e. 'set/unset/show' with Section 2.21.118 [cbrange], page 162, '[m]cbtics', 'format cb', 'grid [m]cb', Section 2.21.116 [cblabel], page 162, and perhaps even Section 2.21.108 [cbdata], page 161, '[no]cbdtics', '[no]cbmtics'.

'set colorbox' without any parameter switches the position to default. 'unset colorbox' resets the default parameters for the colorbox and switches the colorbox off.

See also help for pm3d, Section 2.21.50 [palette], page 120, pm3d, and 'set style line'.

2.21.12 contour

Section 2.21.12 [contour], page 86 enables contour drawing for surfaces. This option is available for 'splot' only.

Syntax:

```
set contour {base | surface | both}
unset contour
show contour
```

The three options specify where to draw the contours: 'base' draws the contours on the grid base where the x/ytics are placed, Section 2.21.59 [surface], page 144 draws the contours on the surfaces themselves, and 'both' draws the contours on both the base and the surface. If no option is provided, the default is 'base'.

See also Section 2.21.10 [cntrparam], page 83 for the parameters that affect the drawing of contours, and Section 2.21.8 [clabel], page 82 for control of labelling of the contours.

The surface can be switched off (see Section 2.21.59 [surface], page 144), giving a contour-only graph. Though it is possible to use Section 2.21.57 [size], page 129 to enlarge the plot to fill the screen, more control over the output format can be obtained by writing the contour information to a file, and rereading it as a 2-d datafile plot:

```
unset surface
set contour
set cntrparam ...
set table 'filename'
splot ...
unset table
# contour info now in filename
set term <whatever>
plot 'filename'
```

In order to draw contours, the data should be organized as "grid data". In such a file all the points for a single y-isoline are listed, then all the points for the next y-isoline, and so on. A single blank line (a line containing no characters other than blank spaces and a carriage return and/or a line feed) separates one y-isoline from the next. See also Section 2.21.14 [datafile], page 86.

If contours are desired from non-grid data, Section 2.21.16 [dgrid3d], page 90 can be used to create an appropriate grid. See Section 2.21.16 [dgrid3d], page 90 for more information. See also contours demo (contours.dem) and contours with user defined levels demo (discrete.dem).

2.21.13 data style

This form of the command is deprecated. Please see 'set style data'.

2.21.14 datafile

The Section 2.21.14 [datafile], page 86 command options control interpretation of fields read from input data files by the 'plot', 'splot', and 'fit' commands. Four such options are currently implemented.

2.21.14.1 set datafile fortran

The ‘set datafile fortran‘ command enables a special check for values in the input file expressed as Fortran D or Q constants. This extra check slows down the input process, and should only be selected if you do in fact have datafiles containing Fortran D or Q constants. The option can be disabled again using ‘unset datafile fortran‘.

2.21.14.2 set datafile missing

The ‘set datafile missing‘ command allows you to tell ‘gnuplot‘ what character string is used in a data file to denote missing data. Exactly how this missing value will be treated depends on the Section 2.12.1.9 [using], page 61 specifier of the ‘plot‘ or ‘splot‘ command.

Syntax:

```
set datafile missing {"<string>"}
show datafile missing
unset datafile
```

Example:

```
# Ignore entries containing IEEE NaN ("Not a Number") code
set datafile missing "NaN"
```

Example:

```
set datafile missing "?"
set style data lines
plot '-'
   1 10
   2 20
   3 ?
   4 40
   5 50
   e
plot '-' using 1:2
   1 10
   2 20
   3 ?
   4 40
   5 50
   e
plot '-' using 1:($2)
   1 10
   2 20
   3 ?
   4 40
   5 50
   e
```

The first ‘plot‘ will recognize only the first datum in the "3 ?" line. It will use the single-datum-on-a-line convention that the line number is "x" and the datum is "y", so the point will be plotted (in this case erroneously) at (2,3).

The second ‘plot‘ will correctly ignore the middle line. The plotted line will connect the points at (2,20) and (4,40).

The third ‘plot‘ will also correctly ignore the middle line, but the plotted line will not connect the points at (2,20) and (4,40).

There is no default character for ‘missing‘, but in many cases any non-parsible string of characters found where a numerical value is expected will be treated as missing data.

2.21.14.3 set datafile separator

The command ‘set datafile separator "<char>"‘ tells ‘gnuplot‘ that data fields in subsequent input files are separated by <char> rather than by whitespace. The most common use is to read in csv (comma-separated value) files written by spreadsheet or database programs. By default data fields are separated by whitespace.

Syntax:

```
set datafile separator {"<char>" | whitespace}
```

Examples:

```
# Input file contains tab-separated fields
set datafile separator "\t"

# Input file contains comma-separated values fields
set datafile separator ","
```

2.21.14.4 set datafile commentschars

The ‘set datafile commentschars‘ command allows you to tell ‘gnuplot‘ what characters are used in a data file to denote comments. Gnuplot will ignore rest of the line behind the specified characters if either of them is the first non-blank character on the line.

Syntax:

```
set datafile commentschars {"<string>"}
show datafile commentschars
unset commentschars
```

Default value of the string is "#!" on VMS and "#" otherwise.

Then, the following line in a data file is completely ignored

```
# 1 2 3 4
```

but the following

```
1 # 3 4
```

produces rather unexpected plot unless

```
set datafile missing '#'
```

is specified as well.

Example:

```
set datafile commentschars "#!%"
```

2.21.14.5 set datafile binary

The 'set datafile binary' command is used to set the defaults when reading binary data files. The syntax matches precisely that used for commands 'plot' and 'splot'. See 'binary' for details about <binary list>.

Syntax:

```
set datafile binary <binary list>
show datafile binary
show datafile
unset datafile
```

Examples:

```
set datafile binary filetype=auto
set datafile binary array=512x512 format="%uchar"
```

2.21.15 decimalsign

The Section 2.21.15 [decimalsign], page 89 command selects a decimal sign for numbers printed into tic labels or Section 2.21.29 [label], page 104 strings.

Syntax:

```
set decimalsign {<value> | locale {"<locale>"}}
unset decimalsign
show decimalsign
```

The argument <value> is a string to be used in place of the usual decimal point. Typical choices include the period, '.', and the comma, ',', but others may be useful, too. If you omit the <value> argument, the decimal separator is not modified from the usual default, which is a period. Unsetting decimalsign has the same effect as omitting <value>.

Example:

Correct typesetting in most European countries requires:

```
set decimalsign ','
```

Please note: If you set an explicit string, this affects only numbers that are printed using gnuplot's gprintf() formatting routine, include axis tics. It does not affect the format expected for input data, and it does not affect numbers printed with the sprintf() formatting routine. To change the behavior of both input and output formatting, instead use the form

```
set decimalsign locale
```

This instructs the program to use both input and output formats in accordance with the current setting of the LC_ALL, LC_NUMERIC, or LANG environmental variables.

```
set decimalsign locale "foo"
```

This instructs the program to format all input and output in accordance with locale "foo", which must be installed. If locale "foo" is not found then an error message is printed and the decimal sign setting is unchanged.

```
set decimalsign locale; set decimalsign "."
```

This sets all input and output to use whatever decimal sign is correct for the current locale, but over-rides this with an explicit '.' in numbers formatted using gnuplot's internal gprintf() function.

2.21.16 dgrid3d

The Section 2.21.16 [dgrid3d], page 90 command enables, and can set parameters for, non-grid to grid data mapping.

Syntax:

```
set dgrid3d {<row_size>} {,{<col_size>} {,<norm>}}
unset dgrid3d
show dgrid3d
```

By default Section 2.21.16 [dgrid3d], page 90 is disabled. When enabled, 3-d data read from a file are always treated as a scattered data set. A grid with dimensions derived from a bounding box of the scattered data and size as specified by the row/col_size parameters is created for plotting and contouring. The grid is equally spaced in x (rows) and in y (columns); the z values are computed as weighted averages of the scattered points' z values.

The third parameter, norm, controls the weighting: Each data point is weighted inversely by its distance from the grid point raised to the norm power. (Actually, the weights are given by the inverse of dx^norm + dy^norm, where dx and dy are the components of the separation of the grid point from each data point. For some norms that are powers of two, specifically 4, 8, and 16, the computation is optimized by using the Euclidean distance in the weight calculation, (dx^2+dy^2)^norm/2. However, any non-negative integer can be used.)

The closer the data point is to a grid point, the more effect it has on that grid point and the larger the value of norm the less effect more distant data points have on that grid point.

The Section 2.21.16 [dgrid3d], page 90 option is a simple low pass filter that converts scattered data to a grid data set. More sophisticated approaches to this problem exist and should be used to preprocess the data outside 'gnuplot' if this simple solution is found inadequate.

(The z values are found by weighting all data points, not by interpolating between nearby data points; also edge effects may produce unexpected and/or undesired results. In some cases, small norm values produce a grid point reflecting the average of distant data points rather than a local average, while large values of norm may produce "steps" with several grid points having the same value as the closest data point, rather than making a smooth transition between adjacent data points. Some areas of a grid may be filled by extrapolation, to an arbitrary boundary condition. The variables are not normalized; consequently the units used for x and y will affect the relative weights of points in the x and y directions.)

Examples:

```
set dgrid3d 10,10,1     # defaults
set dgrid3d ,,4
```

The first specifies that a grid of size 10 by 10 is to be constructed using a norm value of 1 in the weight computation. The second only modifies the norm, changing it to 4. See also scatter.dem: dgrid3d demo.

2.21.17 dummy

The Section 2.21.17 [dummy], page 91 command changes the default dummy variable names.

Syntax:

```
set dummy {<dummy-var>} {,<dummy-var>}
show dummy
```

By default, ‘gnuplot‘ assumes that the independent, or "dummy", variable for the ‘plot‘ command is "t" if in parametric or polar mode, or "x" otherwise. Similarly the independent variables for the ‘splot‘ command are "u" and "v" in parametric mode (‘splot‘ cannot be used in polar mode), or "x" and "y" otherwise.

It may be more convenient to call a dummy variable by a more physically meaningful or conventional name. For example, when plotting time functions:

```
set dummy t
plot sin(t), cos(t)
```

At least one dummy variable must be set on the command; Section 2.21.17 [dummy], page 91 by itself will generate an error message.

Examples:

```
set dummy u,v
set dummy ,s
```

The second example sets the second variable to s.

2.21.18 encoding

The Section 2.21.18 [encoding], page 91 command selects a character encoding. Syntax:

```
set encoding {<value>}
show encoding
```

Valid values are

```
default     - tells a terminal to use its default encoding
iso_8859_1  - the most common Western European font used by many
```

```
                Unix workstations and by MS-Windows. This encoding is
                known in the PostScript world as 'ISO-Latin1'.
iso_8859_2  - used in Central and Eastern Europe
iso_8859_15 - a variant of iso_8859_1 that includes the Euro symbol
koi8r       - popular Unix cyrillic encoding
koi8u       - ukrainian Unix cyrillic encoding
cp437       - codepage for MS-DOS
cp850       - codepage for OS/2, Western Europe
cp852       - codepage for OS/2, Central and Eastern Europe
cp1250      - codepage for MS Windows, Central and Eastern Europe
```

Generally you must set the encoding before setting the terminal type. Note that encoding is not supported by all terminal drivers and that the device must be able to produce the desired non-standard characters. The PostScript and X11 terminals support all encodings. OS/2 Presentation Manager switches automatically to codepage 912 for 'iso_8859_2'.

2.21.19 fit

The 'fit' setting defines where the 'fit' command writes its output. If this option was built into your version of gnuplot, it also controls whether parameter errors from the fit will be written into variables.

Syntax:

```
set fit {logfile {"<filename>"}} {{no}errorvariables}
unset fit
show fit
```

The <filename> argument must be enclosed in single or double quotes.

If no filename is given or 'unset fit' is used the log file is reset to its default value "fit.log" or the value of the environmental variable 'FIT_LOG'.

Users of DOS-like platforms should note that the \ character has special significance in double-quoted strings, so single-quotes should be used for filenames in different directories, or you have to write \\ for each \. Or you can just use forward slashes, even though this is DOS.

If the given logfile name ends with a / or \, it is interpreted to be a directory name, and the actual filename will be "fit.log" in that directory.

If the 'errorvariables' option is turned on, the error of each fitted parameter computed by 'fit' will be copied to a user-defined variable whose name is formed by appending "_err" to the name of the parameter itself. This is useful mainly to put the parameter and its error onto a plot of the data and the fitted function, for reference, as in:

```
set fit errorvariables
fit f(x) 'datafile' using 1:2 via a, b
print "error of a is:", a_err
set label 'a=%6.2f', a, '+/- %6.2f', a_err
plot 'datafile' using 1:2, f(x)
```

2.21.20 fontpath

The Section 2.21.20 [fontpath], page 93 setting defines additional locations for font files searched when including font files. Currently only the postscript terminal supports Section 2.21.20 [fontpath], page 93. If a file cannot be found in the current directory, the directories in Section 2.21.20 [fontpath], page 93 are tried. Further documentation concerning the supported file formats is included in the Section 2.21.50.6 [postscript], page 126 section of the documentation.

Syntax:

```
set fontpath {"pathlist1" {"pathlist2"...}}
show fontpath
```

Path names may be entered as single directory names, or as a list of path names separated by a platform-specific path separator, eg. colon (':') on Unix, semicolon (';') on DOS/Windows/OS/2/Amiga platforms. The Section 2.21.20 [fontpath], page 93, Section 2.20 [save], page 73 and 'save set' commands replace the platform-specific separator with a space character (' ') for maximum portability. If a directory name ends with an exclamation mark ('!') also the subdirectories of this directory are searched for font files.

If the environmental variable GNUPLOT_FONTPATH is set, its contents are appended to Section 2.21.20 [fontpath], page 93. If it is not set, a system dependent default value is used. It is set by testing several directories for existence when using the fontpath the first time. Thus, the first call of Section 2.21.20 [fontpath], page 93, Section 2.21.20 [fontpath], page 93, Section 2.21.20 [fontpath], page 93, 'plot', or 'splot' with embedded font files takes a little more time. If you want to save this time you may set the environmental variable GNUPLOT_FONTPATH since probing is switched off, then. You can find out which is the default fontpath by using Section 2.21.20 [fontpath], page 93.

However, Section 2.21.20 [fontpath], page 93 prints the contents of user defined fontpath and system fontpath separately. Also, the Section 2.20 [save], page 73 and 'save set' commands save only the user specified parts of Section 2.21.20 [fontpath], page 93, for portability reasons.

Many other terminal drivers access TrueType fonts via the gd library. For these drivers the font search path is controlled by the environmental variable GDFONTPATH.

2.21.21 format

The format of the tic-mark labels can be set with the 'set format' command.

Syntax:

```
set format {<axes>} {"<format-string>"}
set format {<axes>} {'<format-string>'}
show format
```

where <axes> is either 'x', 'y', 'z', 'cb', 'xy', 'x2', 'y2' or nothing (which is the same as 'xy'). The length of the string representing a tic mark (after formatting with 'printf') is restricted to 100 characters. If the format string is omitted, the format will be returned to the default "% g". For LaTeX users, the format "$%g$" is often desirable. If the empty string "" is used, no label will be plotted with each tic, though the tic mark will still be plotted. To eliminate all tic marks, use Section 2.21.88 [xtics], page 155 or Section 2.21.103 [ytics], page 160.

Newline (\n) is accepted in the format string. Use double-quotes rather than single-quotes to enable such interpretation. See also ‘syntax‘.

The default format for both axes is "% g", but other formats such as "%.2f" or "%3.0em" are often desirable. Anything accepted by 'printf' when given a double precision number, and accepted by the terminal, will work. Some other options have been added. If the format string looks like a floating point format, then ‘gnuplot‘ tries to construct a reasonable format.

Characters not preceded by "%" are printed verbatim. Thus you can include spaces and labels in your format string, such as "%g m", which will put " m" after each number. If you want "%" itself, double it: "%g %%".

See also Section 2.21.88 [xtics], page 155 for more information about tic labels, and Section 2.21.15 [decimalsign], page 89 for how to use non-default decimal separators in numbers printed this way. See also electron demo (electron.dem).

2.21.21.1 gprintf

The string function gprintf("format",x) uses gnuplot's own format specifiers, as do the gnuplot commands ‘set format‘, Section 2.21.66 [timestamp], page 147, and others. These format specifiers are not the same as those used by the standard C-language routine sprintf(). Gnuplot also provides an sprintf("format",x,...) routine if you prefer. For a list of gnuplot's format options, see ‘format specifiers‘.

2.21.21.2 format specifiers

The acceptable formats (if not in time/date mode) are:

```
      Format       Explanation
      %f           floating point notation
      %e or %E     exponential notation; an "e" or "E" be-
  fore the power
      %g or %G     the shorter of %e (or %E) and %f
      %x or %X     hex
      %o or %O     octal
      %t           mantissa to base 10
      %l           mantissa to base of current logscale
      %s           mantissa to base of current logscale; scien-
  tific power
      %T           power to base 10
      %L           power to base of current logscale
      %S           scientific power
      %c           character replacement for scientific power
      %P           multiple of pi
```

A 'scientific' power is one such that the exponent is a multiple of three. Character replacement of scientific powers (‘"%c"‘) has been implemented for powers in the range -18 to +18. For numbers outside of this range the format reverts to exponential.

Other acceptable modifiers (which come after the "%" but before the format specifier) are "-", which left-justifies the number; "+", which forces all numbers to be explicitly signed; " " (a space), which makes positive numbers have a space in front of them where negative numbers have "-"; "#", which places a decimal point after floats that have only zeroes following the decimal point; a positive integer, which defines the field width; "0" (the digit, not the letter) immediately preceding the field width, which indicates that leading zeroes are to be used instead of leading blanks; and a decimal point followed by a non-negative integer, which defines the precision (the minimum number of digits of an integer, or the number of digits following the decimal point of a float).

Some systems may not support all of these modifiers but may also support others; in case of doubt, check the appropriate documentation and then experiment.

Examples:

```
set format y "%t"; set ytics (5,10)          # "5.0" and "1.0"
set format y "%s"; set ytics (500,1000)      # "500" and "1.0"
set format y "+-12.3f"; set ytics(12345)     # "+12345.000  "
set format y "%.2t*10^%+03T"; set ytic(12345)# "1.23*10^+04"
set format y "%s*10^{%S}"; set ytic(12345)   # "12.345*10^{3}"
set format y "%s %cg"; set ytic(12345)       # "12.345 kg"
set format y "%.0P pi"; set ytic(6.283185)   # "2 pi"
set format y "%.0f%%"; set ytic(50)          # "50%"

set log y 2; set format y '%l'; set ytics (1,2,3)
#displays "1.0", "1.0" and "1.5" (since 3 is 1.5 * 2^1)
```

There are some problem cases that arise when numbers like 9.999 are printed with a format that requires both rounding and a power.

If the data type for the axis is time/date, the format string must contain valid codes for the 'strftime' function (outside of 'gnuplot', type "man strftime"). See Section 2.21.67 [timefmt], page 147 for a list of the allowed input format codes.

2.21.21.3 time/date specifiers

In time/date mode, the acceptable formats are:

```
Format       Explanation
%a           abbreviated name of day of the week
%A           full name of day of the week
%b or %h     abbreviated name of the month
%B           full name of the month
%d           day of the month, 1--31
%D           shorthand for "%m/%d/%y"
%k           hour, 0--23 (one or two digits)
%H           hour, 00--23 (always two digits)
*l           hour, 1--12 (one or two digits)
%I           hour, 01--12 (always two digits)
%j           day of the year, 1--366
%m           month, 1--12
```

```
%M            minute, 0--60
%p            "am" or "pm"
%r            shorthand for "%I:%M:%S %p"
%R            shorthand for "%H:%M"
%S            second, 0--60
%T            shorthand for "%H:%M:%S"
%U            week of the year (week starts on Sunday)
%w            day of the week, 0--6 (Sunday = 0)
%W            week of the year (week starts on Monday)
%y            year, 0-99
%Y            year, 4-digit
```

Except for the non-numerical formats, these may be preceded by a "0" ("zero", not "oh") to pad the field length with leading zeroes, and a positive digit, to define the minimum field width (which will be overridden if the specified width is not large enough to contain the number). There is a 24-character limit to the length of the printed text; longer strings will be truncated.

Examples:

Suppose the text is "76/12/25 23:11:11". Then

```
set format x                 # defaults to "12/25/76" \n "23:11"
set format x "%A, %d %b %Y"  # "Saturday, 25 Dec 1976"
set format x "%r %D"         # "11:11:11 pm 12/25/76"
```

Suppose the text is "98/07/06 05:04:03". Then

```
set for-
mat x "%1y/%2m/%3d %01H:%02M:%03S"  # "98/ 7/  6 5:04:003"
```

2.21.22 function style

This form of the command is deprecated. Please see 'set style function'.

2.21.23 functions

The 'show functions' command lists all user-defined functions and their definitions.

Syntax:

```
show functions
```

For information about the definition and usage of functions in 'gnuplot', please see 'expressions'. See also splines as user defined functions (spline.dem) and use of functions and complex variables for airfoils (airfoil.dem).

2.21.24 grid

The 'set grid' command allows grid lines to be drawn on the plot.

Syntax:

```
set grid {{no}{m}xtics} {{no}{m}ytics} {{no}{m}ztics}
         {{no}{m}x2tics} {{no}{m}y2tics}
         {{no}{m}cbtics}
         {polar {<angle>}}
         {layerdefault | front | back}
         { {linestyle <major_linestyle>}
           | {linetype | lt <major_linetype>}
             {linewidth | lw <major_linewidth>}
           { , {linestyle | ls <minor_linestyle>}
               | {linetype | lt <minor_linetype>}
                 {linewidth | lw <minor_linewidth>} } }
unset grid
show grid
```

The grid can be enabled and disabled for the major and/or minor tic marks on any axis, and the linetype and linewidth can be specified for major and minor grid lines, also via a predefined linestyle, as far as the active terminal driver supports this.

Additionally, a polar grid can be selected for 2-d plots—circles are drawn to intersect the selected tics, and radial lines are drawn at definable intervals. (The interval is given in degrees or radians, depending on the Section 2.21.1 [angles], page 74 setting.) Note that a polar grid is no longer automatically generated in polar mode.

The pertinent tics must be enabled before ‘set grid‘ can draw them; ‘gnuplot‘ will quietly ignore instructions to draw grid lines at non-existent tics, but they will appear if the tics are subsequently enabled.

If no linetype is specified for the minor gridlines, the same linetype as the major gridlines is used. The default polar angle is 30 degrees.

If ‘front‘ is given, the grid is drawn on top of the graphed data. If ‘back‘ is given, the grid is drawn underneath the graphed data. Using ‘front‘ will prevent the grid from being obscured by dense data. The default setup, ‘layerdefault‘, is equivalent to ‘back‘ for 2d plots. In 3D plots the default is to split up the grid and the graph box into two layers: one behind, the other in front of the plotted data and functions. Since Section 2.21.25 [hidden3d], page 97 mode does its own sorting, it ignores all grid drawing order options and passes the grid lines through the hidden line removal machinery instead. These options actually affect not only the grid, but also the lines output by Section 2.21.6 [border], page 79 and the various ticmarks (see Section 2.21.88 [xtics], page 155).

Z grid lines are drawn on the bottom of the plot. This looks better if a partial box is drawn around the plot—see Section 2.21.6 [border], page 79.

2.21.25 hidden3d

The Section 2.21.25 [hidden3d], page 97 command enables hidden line removal for surface plotting (see ‘splot‘). Some optional features of the underlying algorithm can also be controlled using this command.

Syntax:

```
set hidden3d {defaults} |
             { {{offset <offset>} | {nooffset}}
```

```
                              {trianglepattern <bitpattern>}
                              {{undefined <level>} | {noundefined}}
                              {{no}altdiagonal}
                              {{no}bentover} }
       unset hidden3d
       show hidden3d
```

In contrast to the usual display in gnuplot, hidden line removal actually treats the given function or data grids as real surfaces that can't be seen through, so parts behind the surface will be hidden by it. For this to be possible, the surface needs to have 'grid structure' (see Section 2.21.14 [datafile], page 86 about this), and it has to be drawn 'with lines' or 'with linespoints'.

When Section 2.21.25 [hidden3d], page 97 is set, both the hidden portion of the surface and possibly its contours drawn on the base (see Section 2.21.12 [contour], page 86) as well as the grid will be hidden. Each surface has its hidden parts removed with respect to itself and to other surfaces, if more than one surface is plotted. Contours drawn on the surface (Section 2.21.59 [surface], page 144) don't work.

Labels and arrows are always visible and are unaffected. The key box is never hidden by the surface. As of gnuplot version 4.1, Section 2.21.25 [hidden3d], page 97 also affects 3D plotting styles 'with points', 'with labels', and 'with vectors', even if no surface is present in the graph. Individual plots within the graph may be explicitly excluded from this processing by appending the extra option 'nohidden3d' to the Section 2.12.7 [with], page 68 specifier.

Functions are evaluated at isoline intersections. The algorithm interpolates linearly between function points or data points when determining the visible line segments. This means that the appearance of a function may be different when plotted with Section 2.21.25 [hidden3d], page 97 than when plotted with 'nohidden3d' because in the latter case functions are evaluated at each sample. Please see Section 2.21.56 [samples], page 128 and Section 2.21.27 [isosamples], page 100 for discussion of the difference.

The algorithm used to remove the hidden parts of the surfaces has some additional features controllable by this command. Specifying 'defaults' will set them all to their default settings, as detailed below. If 'defaults' is not given, only explicitly specified options will be influenced: all others will keep their previous values, so you can turn on/off hidden line removal via 'set {no}hidden3d', without modifying the set of options you chose.

The first option, 'offset', influences the linestyle used for lines on the 'back' side. Normally, they are drawn in a linestyle one index number higher than the one used for the front, to make the two sides of the surface distinguishable. You can specify a different line style offset to add instead of the default 1, by 'offset <offset>'. Option 'nooffset' stands for 'offset 0', making the two sides of the surface use the same linestyle.

Next comes the option 'trianglepattern <bitpattern>'. <bitpattern> must be a number between 0 and 7, interpreted as a bit pattern. Each bit determines the visibility of one edge of the triangles each surface is split up into. Bit 0 is for the 'horizontal' edges of the grid, Bit 1 for the 'vertical' ones, and Bit 2 for the diagonals that split each cell of the original grid into two triangles. The default pattern is 3, making all horizontal and vertical lines visible, but not the diagonals. You may want to choose 7 to see those diagonals as well.

The ‘undefined <level>‘ option lets you decide what the algorithm is to do with data points that are undefined (missing data, or undefined function values), or exceed the given x-, y- or z-ranges. Such points can either be plotted nevertheless, or taken out of the input data set. All surface elements touching a point that is taken out will be taken out as well, thus creating a hole in the surface. If <level> = 3, equivalent to option ‘noundefined‘, no points will be thrown away at all. This may produce all kinds of problems elsewhere, so you should avoid this. <level> = 2 will throw away undefined points, but keep the out-of-range ones. <level> = 1, the default, will get rid of out-of-range points as well.

By specifying ‘noaltdiagonal‘, you can override the default handling of a special case can occur if ‘undefined‘ is active (i.e. <level> is not 3). Each cell of the grid-structured input surface will be divided in two triangles along one of its diagonals. Normally, all these diagonals have the same orientation relative to the grid. If exactly one of the four cell corners is excluded by the ‘undefined‘ handler, and this is on the usual diagonal, both triangles will be excluded. However if the default setting of ‘altdiagonal‘ is active, the other diagonal will be chosen for this cell instead, minimizing the size of the hole in the surface.

The ‘bentover‘ option controls what happens to another special case, this time in conjunction with the ‘trianglepattern‘. For rather crumply surfaces, it can happen that the two triangles a surface cell is divided into are seen from opposite sides (i.e. the original quadrangle is ’bent over’), as illustrated in the following ASCII art:

```
                                                          C----B
  original quadrangle:  A--B      displayed quadrangle:   |\   |
    ("set view 0,0")    | /|    ("set view 75,75" perhaps) | \  |
                        |/ |                              |  \ |
                        C--D                              |   \|
                                                          A    D
```

If the diagonal edges of the surface cells aren’t generally made visible by bit 2 of the <bitpattern> there, the edge CB above wouldn’t be drawn at all, normally, making the resulting display hard to understand. Therefore, the default option of ‘bentover‘ will turn it visible in this case. If you don’t want that, you may choose ‘nobentover‘ instead. See also hidden line removal demo (hidden.dem) and complex hidden line demo (singulr.dem).

2.21.26 historysize

Note: the command Section 2.21.26 [historysize], page 99 is only available when gnuplot has been configured with the GNU readline.

Syntax:

```
set historysize <int>
unset historysize
```

When leaving gnuplot, the value of historysize is used for truncating the history to at most that much lines. The default is 500. Section 2.21.26 [historysize], page 99 will disable history truncation and thus allow an infinite number of lines to be written to the history file.

2.21.27 isosamples

The isoline density (grid) for plotting functions as surfaces may be changed by the Section 2.21.27 [isosamples], page 100 command.

Syntax:

```
set isosamples <iso_1> {,<iso_2>}
show isosamples
```

Each function surface plot will have <iso_1> iso-u lines and <iso_2> iso-v lines. If you only specify <iso_1>, <iso_2> will be set to the same value as <iso_1>. By default, sampling is set to 10 isolines per u or v axis. A higher sampling rate will produce more accurate plots, but will take longer. These parameters have no effect on data file plotting.

An isoline is a curve parameterized by one of the surface parameters while the other surface parameter is fixed. Isolines provide a simple means to display a surface. By fixing the u parameter of surface s(u,v), the iso-u lines of the form c(v) = s(u0,v) are produced, and by fixing the v parameter, the iso-v lines of the form c(u) = s(u,v0) are produced.

When a function surface plot is being done without the removal of hidden lines, Section 2.21.56 [samples], page 128 controls the number of points sampled along each isoline; see Section 2.21.56 [samples], page 128 and Section 2.21.25 [hidden3d], page 97. The contour algorithm assumes that a function sample occurs at each isoline intersection, so change in Section 2.21.56 [samples], page 128 as well as Section 2.21.27 [isosamples], page 100 may be desired when changing the resolution of a function surface/contour.

2.21.28 key

The Section 2.21.28 [key], page 100 enables a key (or legend) describing plots on a plot.

The contents of the key, i.e., the names given to each plotted data set and function and samples of the lines and/or symbols used to represent them, are determined by the ‘title‘ and Section 2.12.7 [with], page 68 options of the {‘s‘}‘plot‘ command. Please see ‘plot title‘ and Section 2.12.7 [with], page 68 for more information.

Syntax:

```
        set key {on|off} {default}
                {{inside | outside} | {lmargin | rmargin | tmar-
  gin | bmargin}
                  | {at <position>}}
                {left | right | center} {top | bottom | center}
                {vertical | horizontal} {Left | Right}
                {{no}reverse} {{no}invert}
                {samplen <sample_length>} {spacing <vertical_spacing>}
                {width <width_increment>}
                {height <height_increment>}
                {{no}autotitle {columnheader}}
                {title "<text>"} {{no}enhanced}
                {{no}box { {linestyle | ls <line_style>}
                           | {linetype | lt <line_type>}
                             {linewidth | lw <line_width>}}}
        unset key
```

```
show key
```

Plots may be drawn with no visible key by requesting ‘set key off‘ or Section 2.21.28 [key], page 100.

Elements within the key are stacked according to ‘vertical‘ or ‘horizontal‘. In the case of ‘vertical‘, the key occupies as few columns as possible. That is, elements are aligned in a column until running out of vertical space at which point a new column is started. In the case of ‘horizontal‘, the key occupies as few rows as possible.

By default the key is placed in the upper right inside corner of the graph. The keywords ‘left‘, ‘right‘, ‘top‘, ‘bottom‘, ‘center‘, ‘inside‘, ‘outside‘, Section 2.21.30 [lmargin], page 106, Section 2.21.54 [rmargin], page 128, Section 2.21.69 [tmargin], page 149, Section 2.21.5 [bmargin], page 79 (, ‘above‘, ‘over‘, ‘below‘ and ‘under‘) may be used to automatically place the key in other positions of the graph. Also an ‘at `<position>`‘ may be given to indicate precisely where the plot should be placed. In this case, the keywords ‘left‘, ‘right‘, ‘center‘, ‘bottom‘ and ‘center‘ serve an analogous purpose for alignment.

To understand positioning, the best concept is to think of a region, i.e., inside/outside, or one of the margins. Along with the region, keywords ‘left/center/right‘ (l/c/r) and ‘top/center/bottom‘ (t/c/b) control where within the particular region the key should be placed.

When in ‘inside‘ mode, the keywords ‘left‘ (l), ‘right‘ (r), ‘top‘ (t), ‘bottom‘ (b), and ‘center‘ (c) push the key out toward the plot boundary as illustrated:

```
t/l   t/c   t/r


c/l    c    c/r


b/l   b/c   b/r
```

When in ‘outside‘ mode, automatic placement is similar to the above illustration, but with respect to the view, rather than the graph boundary. That is, a border is moved inward to make room for the key outside of the plotting area, although this may interfere with other labels and may cause an error on some devices. The particular plot border that is moved depends upon the position described above and the stacking direction. For options centered in one of the dimensions, there is no ambiguity about which border to move. For the corners, when the stack direction is ‘vertical‘, the left or right border is moved inward appropriately. When the stack direction is ‘horizontal‘, the top or bottom border is moved inward appropriately.

The margin syntax allows automatic placement of key regardless of stack direction. When one of the margins Section 2.21.30 [lmargin], page 106 (lm), Section 2.21.54 [rmargin], page 128 (rm), Section 2.21.69 [tmargin], page 149 (tm), and Section 2.21.5 [bmargin], page 79 (bm) is combined with a single, non-conflicting direction keyword, the following illustrated positions may contain the key:

```
     l/tm  c/tm  r/tm


t/lm                  t/rm
```

```
    c/lm                    c/rm


    b/lm                    b/rm


         l/bm  c/bm  r/bm
```

Keywords ‘above‘ and ‘over‘ are synonymous with Section 2.21.69 [tmargin], page 149. For version compatibility, ‘above‘ or ‘over‘ without an additional l/c/r or stack direction keyword uses ‘center‘ and ‘horizontal‘. Keywords ‘below‘ and ‘under‘ are synonymous with Section 2.21.5 [bmargin], page 79. For compatibility, ‘below‘ or ‘under‘ without an additional l/c/r or stack direction keyword uses ‘center‘ and ‘horizontal‘. A further compatibility issue is that ‘outside‘ appearing without an additional t/b/c or stack direction keyword uses ‘top‘, ‘right‘ and ‘vertical‘ (i.e., the same as t/rm above).

The <position> can be a simple x,y,z as in previous versions, but these can be preceded by one of five keywords (‘first‘, ‘second‘, ‘graph‘, ‘screen‘, ‘character‘) which selects the coordinate system in which the position of the first sample line is specified. See ‘coordinates‘ for more details. The effect of ‘left‘, ‘right‘, ‘top‘, ‘bottom‘, and ‘center‘ when <position> is given is to align the key as though it were text positioned using the label command, i.e., ‘left‘ means left align with key to the right of <position>, etc.

Justification of the labels within the key is controlled by ‘Left‘ or ‘Right‘ (default is ‘Right‘). The text and sample can be reversed (‘reverse‘) and a box can be drawn around the key (‘box {...}‘) in a specified ‘linetype‘ and ‘linewidth‘, or a user-defined ‘linestyle‘. Note that not all terminal drivers support linewidth selection, though.

By default the first plot label is at the top of the key and successive labels are entered below it. The ‘invert‘ option causes the first label to be placed at the bottom of the key, with successive labels entered above it. This option is useful to force the vertical ordering of labels in the key to match the order of box types in a stacked histogram.

The length of the sample line can be controlled by ‘samplen‘. The sample length is computed as the sum of the tic length and <sample_length> times the character width. ‘samplen‘ also affects the positions of point samples in the key since these are drawn at the midpoint of the sample line, even if the sample line itself is not drawn.

The vertical spacing between lines is controlled by ‘spacing‘. The spacing is set equal to the product of the pointsize, the vertical tic size, and <vertical_spacing>. The program will guarantee that the vertical spacing is no smaller than the character height.

The <width_increment> is a number of character widths to be added to or subtracted from the length of the string. This is useful only when you are putting a box around the key and you are using control characters in the text. ‘gnuplot‘ simply counts the number of characters in the string when computing the box width; this allows you to correct it.

The <height_increment> is a number of character heights to be added to or subtracted from the height of the key box. This is useful mainly when you are putting a box around the key, otherwise it can be used to adjust the vertical shift of automatically chosen key position by <height_increment>/2.

All plotted curves of ‘plot‘s and ‘splot‘s are titled according to the default option ‘autotitles‘. The automatic generation of titles can be suppressed by ‘noautotitles‘; then only those titles explicitly defined by ‘(s)plot ... title ...‘ will be drawn.

The ‘set key autotitle columnheader‘ option is available if gnuplot was built with –enable-datastrings. This command causes the first entry in each column of plotted data to be interpreted as a text string and used as a title for the corresponding plot. If the quantity being plotted is a function of data from several columns, gnuplot may be confused as to which column to draw the title from. In this case it is necessary to specify the column explicitly in the plot command, e.g. ‘plot "datafile" using (($2+$3)/$4) title 3 with lines‘.

A title can be put on the key (‘title "<text>"‘)—see also ‘syntax‘ for the distinction between text in single- or double-quotes. The key title uses the same justification as do the plot titles.

An explicitly given title is typeset using enhanced text properties on terminals supporting this, see ‘enhanced text‘ for more details. This default behavior can be switched off by the ‘noenhanced‘ option.

The defaults for Section 2.21.28 [key], page 100 are ‘on‘, ‘right‘, ‘top‘, ‘vertical‘, ‘Right‘, ‘noreverse‘, ‘noinvert‘, ‘samplen 4‘, ‘spacing 1.25‘, ‘title ""‘, and ‘nobox‘. The default <linetype> is the same as that used for the plot borders. Entering ‘set key default‘ returns the key to its default configuration.

The key is drawn as a sequence of lines, with one plot described on each line. On the right-hand side (or the left-hand side, if ‘reverse‘ is selected) of each line is a representation that attempts to mimic the way the curve is plotted. On the other side of each line is the text description (the line title), obtained from the ‘plot‘ command. The lines are vertically arranged so that an imaginary straight line divides the left- and right-hand sides of the key. It is the coordinates of the top of this line that are specified with the Section 2.21.28 [key], page 100 command. In a ‘plot‘, only the x and y coordinates are used to specify the line position. For a ‘splot‘, x, y and z are all used as a 3-d location mapped using the same mapping as the graph itself to form the required 2-d screen position of the imaginary line.

When using the TeX or PostScript drivers, or similar drivers where formatting information is embedded in the string, ‘gnuplot‘ is unable to calculate correctly the width of the string for key positioning. If the key is to be positioned at the left, it may be convenient to use the combination ‘set key left Left reverse‘. The box and gap in the grid will be the width of the literal string.

If ‘splot‘ is being used to draw contours, the contour labels will be listed in the key. If the alignment of these labels is poor or a different number of decimal places is desired, the label format can be specified. See Section 2.21.8 [clabel], page 82 for details.

Examples:

This places the key at the default location:

```
set key default
```

This disables the key:

```
unset key
```

This places a key at coordinates 2,3.5,2 in the default (first) coordinate system:

```
set key 2,3.5,2
```

This places the key below the graph:

```
set key below
```

This places the key in the bottom left corner, left-justifies the text, gives it a title, and draws a box around it in linetype 3:

```
set key left bottom Left title 'Legend' box 3
```

2.21.29 label

Arbitrary labels can be placed on the plot using the Section 2.21.29 [label], page 104 command.

Syntax:

```
set label {<tag>} {"<label text>"} {at <position>}
          {left | center | right}
          {norotate | rotate {by <degrees>}}
          {font "<name>{,<size>}"}
          {noenhanced}
          {front | back}
          {textcolor <colorspec>}
          {point <pointstyle> | nopoint}
          {offset <offset>}
unset label {<tag>}
show label
```

The <position> is specified by either x,y or x,y,z, and may be preceded by 'first', 'second', 'graph', 'screen', or 'character' to select the coordinate system. See 'coordinates' for details.

The tag is an integer that is used to identify the label. If no <tag> is given, the lowest unused tag value is assigned automatically. The tag can be used to delete or modify a specific label. To change any attribute of an existing label, use the Section 2.21.29 [label], page 104 command with the appropriate tag, and specify the parts of the label to be changed.

The <label text> can be a string constant, a string variable, or a string- valued expression. See 'strings', Section 1.12.1.42 [sprintf], page 17, and 'gprintf'.

By default, the text is placed flush left against the point x,y,z. To adjust the way the label is positioned with respect to the point x,y,z, add the justification parameter, which may be 'left', 'right' or 'center', indicating that the point is to be at the left, right or center of the text. Labels outside the plotted boundaries are permitted but may interfere with axis labels or other text.

If 'rotate' is given, the label is written vertically (if the terminal can do so, of course). If 'rotate by <degrees>' is given, conforming terminals will try to write the text at the specified angle; non-conforming terminals will treat this as vertical text.

Font and its size can be chosen explicitly by 'font "<name>{,<size>}"' if the terminal supports font settings. Otherwise the default font of the terminal will be used.

Normally the enhanced text mode string interpretation, if enabled for the current terminal, is applied to all text strings including label text. The ‘noenhanced‘ property can be used to exempt a specific label from the enhanced text mode processing. The can be useful if the label contains underscores, for example. See ‘enhanced text‘.

If ‘front‘ is given, the label is written on top of the graphed data. If ‘back‘ is given (the default), the label is written underneath the graphed data. Using ‘front‘ will prevent a label from being obscured by dense data.

‘Textcolor <colorspec>‘ changes the color of the label text. ‘<colorspec>‘ is either a linetype or a mapping onto the pm3d color palette (available only in ‘splot‘), see help for Section 2.21.50 [palette], page 120.

```
   ‘textcolor‘ may be abbreviated ‘tc‘.
   ‘tc default‘ resets the text color to its default state.
   ‘tc lt <n>‘ sets the text color to that of line type <n>.
   ‘tc ls <n>‘ sets the text color to that of line style <n>.
   ‘tc palette z‘ selects a palette color corresponding to the la-
bel z position.
   ‘tc palette cb <val>‘ selects a color correspond-
ing to <val> on the colorbar.
   ‘tc palette fraction <val>‘, with 0<=val<=1, selects a color corre-
sponding to
       the mapping [0:1] to grays/colors of the Section 2.21.50
[palette], page 120.
```

If a <pointstyle> is given, using keywords ‘lt‘, ‘pt‘ and ‘ps‘, see Section 2.21.58 [style], page 130, a point with the given style and color of the given line type is plotted at the label position and the text of the label is displaced slightly. This option is used by default for placing labels in ‘mouse‘ enhanced terminals. Use ‘nopoint‘ to turn off the drawing of a point near the label (this is the default).

The displacement defaults to 1,1 in Section 2.21.51 [pointsize], page 126 units if a <pointstyle> is given, 0,0 if no <pointstyle> is given. The displacement can be controlled by the optional ‘offset <offset>‘ where <offset> is specified by either x,y or x,y,z, and may be preceded by ‘first‘, ‘second‘, ‘graph‘, ‘screen‘, or ‘character‘ to select the coordinate system. See ‘coordinates‘ for details.

If one (or more) axis is timeseries, the appropriate coordinate should be given as a quoted time string according to the Section 2.21.67 [timefmt], page 147 format string. See Section 2.21.83 [xdata], page 151 and Section 2.21.67 [timefmt], page 147.

The EEPIC, Imagen, LaTeX, and TPIC drivers allow \\ in a string to specify a newline.

Examples:

To set a label at (1,2) to "y=x", use:

```
set label "y=x" at 1,2
```

To set a Sigma of size 24, from the Symbol font set, at the center of the graph, use:

```
set label "S" at graph 0.5,0.5 center font "Symbol,24"
```

To set a label "y=x^2" with the right of the text at (2,3,4), and tag the label as number 3, use:

```
set label 3 "y=x^2" at 2,3,4 right
```

To change the preceding label to center justification, use:

```
set label 3 center
```

To delete label number 2, use:

```
unset label 2
```

To delete all labels, use:

```
unset label
```

To show all labels (in tag order), use:

```
show label
```

To set a label on a graph with a timeseries on the x axis, use, for example:

```
set timefmt "%d/%m/%y,%H:%M"
set label "Harvest" at "25/8/93",1
```

To display a freshly fitted parameter on the plot with the data and the fitted function, do this after the 'fit', but before the 'plot':

```
set label sprintf("a = %3.5g",par_a) at 30,15
bfit = gprintf("b = %s*10^%S",par_b)
set label bfit at 30,20
```

To set a label displaced a little bit from a small point:

```
set label 'origin' at 0,0 point lt 1 pt 2 ps 3 offset 1,-1
```

To set a label whose color matches the z value (in this case 5.5) of some point on a 3D splot colored using pm3d:

```
set label 'text' at 0,0,5.5 tc palette z
```

2.21.30 lmargin

The command Section 2.21.30 [lmargin], page 106 sets the size of the left margin. Please see Section 2.21.36 [margin], page 109 for details.

2.21.31 loadpath

The Section 2.21.31 [loadpath], page 106 setting defines additional locations for data and command files searched by the Section 2.2 [call], page 35, 'load', 'plot' and 'splot' commands. If a file cannot be found in the current directory, the directories in Section 2.21.31 [loadpath], page 106 are tried.

Syntax:

```
set loadpath {"pathlist1" {"pathlist2"...}}
show loadpath
```

Path names may be entered as single directory names, or as a list of path names separated by a platform-specific path separator, eg. colon (':') on Unix, semicolon (';') on DOS/Windows/OS/2/Amiga platforms. The Section 2.21.31 [loadpath], page 106, Section 2.20 [save], page 73 and 'save set' commands replace the platform-specific separator with a space character (' ') for maximum portability.

If the environment variable GNUPLOT_LIB is set, its contents are appended to Section 2.21.31 [loadpath], page 106. However, Section 2.21.31 [loadpath], page 106 prints the contents of user defined loadpath and system loadpath separately. Also, the Section 2.20 [save], page 73 and 'save set' commands save only the user specified parts of Section 2.21.31 [loadpath], page 106, for portability reasons.

2.21.32 locale

The Section 2.21.32 [locale], page 107 setting determines the language with which '{x,y,z}{d,m}tics' will write the days and months.

Syntax:

```
set locale {"<locale>"}
```

<locale> may be any language designation acceptable to your installation. See your system documentation for the available options. The default value is determined from the LC_TIME, LC_ALL, or LANG environment variables.

To change the decimal point locale, see Section 2.21.15 [decimalsign], page 89.

2.21.33 logscale

Log scaling may be set on the x, y, z, x2 and/or y2 axes.

Syntax:

```
set logscale <axes> <base>
unset logscale <axes>
show logscale
```

where <axes> may be any combinations of 'x', 'y', 'z', and 'cb' in any order, or 'x2' or 'y2' and where <base> is the base of the log scaling. If <base> is not given, then 10 is assumed. If <axes> is not given, then all axes are assumed. Section 2.21.33 [logscale], page 107 turns off log scaling for the specified axes.

Examples:

To enable log scaling in both x and z axes:

```
set logscale xz
```

To enable scaling log base 2 of the y axis:

```
set logscale y 2
```

To enable z and color log axes for a pm3d plot:

```
set logscale zcb
```

To disable z axis log scaling:

```
unset logscale z
```

2.21.34 macros

If command line macro substitution is enabled, then tokens in the command line of the form @<stringvariablename> will be replaced by the text string contained in <stringvariablename>. See ‘substitution‘.

Syntax:

```
set macros
```

2.21.35 mapping

If data are provided to ‘splot‘ in spherical or cylindrical coordinates, the Section 2.21.35 [mapping], page 108 command should be used to instruct ‘gnuplot‘ how to interpret them.

Syntax:

```
set mapping {cartesian | spherical | cylindrical}
```

A cartesian coordinate system is used by default.

For a spherical coordinate system, the data occupy two or three columns (or Section 2.12.1.9 [using], page 61 entries). The first two are interpreted as the azimuthal and polar angles theta and phi (or "longitude" and "latitude"), in the units specified by Section 2.21.1 [angles], page 74. The radius r is taken from the third column if there is one, or is set to unity if there is no third column. The mapping is:

```
x = r * cos(theta) * cos(phi)
y = r * sin(theta) * cos(phi)
z = r * sin(phi)
```

Note that this is a "geographic" spherical system, rather than a "polar" one (that is, phi is measured from the equator, rather than the pole).

For a cylindrical coordinate system, the data again occupy two or three columns. The first two are interpreted as theta (in the units specified by Section 2.21.1 [angles], page 74) and z. The radius is either taken from the third column or set to unity, as in the spherical case. The mapping is:

```
x = r * cos(theta)
y = r * sin(theta)
z = z
```

The effects of Section 2.21.35 [mapping], page 108 can be duplicated with the Section 2.12.1.9 [using], page 61 filter on the 'splot' command, but Section 2.21.35 [mapping], page 108 may be more convenient if many data files are to be processed. However even if Section 2.21.35 [mapping], page 108 is used, Section 2.12.1.9 [using], page 61 may still be necessary if the data in the file are not in the required order.

Section 2.21.35 [mapping], page 108 has no effect on 'plot'. world.dem: mapping demos.

2.21.36 margin

The computed margins can be overridden by the Section 2.21.36 [margin], page 109 commands. Section 2.21.36 [margin], page 109 shows the current settings.

Syntax:

```
set bmargin {<margin>}
set lmargin {<margin>}
set rmargin {<margin>}
set tmargin {<margin>}
show margin
```

The units of <margin> are character heights or widths, as appropriate. A positive value defines the absolute size of the margin. A negative value (or none) causes 'gnuplot' to revert to the computed value. For 3D plots, only the left margin setting has any effect so far.

Normally the margins of a plot are automatically calculated based on tics, tic labels, axis labels, the plot title, the timestamp and the size of the key if it is outside the borders. If, however, tics are attached to the axes ('set xtics axis', for example), neither the tics themselves nor their labels will be included in either the margin calculation or the calculation of the positions of other text to be written in the margin. This can lead to tic labels overwriting other text if the axis is very close to the border.

2.21.37 mouse

The command 'set mouse' enables mouse actions. Currently the pm, x11, ggi and windows terminals are mouse enhanced. There are two mouse modes. The 2d-graph mode works for 2d graphs and for maps (i.e. splots with Section 2.21.74 [view], page 150 having z-rotation 0, 90, 180, 270 or 360 degrees, including 'set view map') and it allows tracing the position over graph, zooming, annotating graph etc. For 3d graphs 'splot', the view and scaling of the graph can be changed with mouse buttons 1 and 2. If additionally to these buttons the modifier <ctrl> is hold down, the coordinate system only is rotated which is useful for large data sets. A vertical motion of Button 2 with the shift key hold down changes the Section 2.21.64 [ticslevel], page 147.

Mousing is not available in multiplot mode. When multiplot is finished using Section 2.21.38 [multiplot], page 111, then the mouse will be turned on again and acts on the last plot (like replot does).

Syntax:

```
set mouse [doubleclick <ms>] [nodoubleclick] \
          [[no]zoomcoordinates] \
          [[no]polardistance] \
```

```
              [format <string>] \
              [clipboardformat <int>/<string>] \
              [mouseformat <int>/<string>] \
              [[no]labels] [labeloptions <string>] \
              [[no]zoomjump] [[no]verbose]
unset mouse
```

The doubleclick resolution is given in milliseconds and used for Button 1 which copies the current mouse position to the ‘clipboard‘. If you want that to be done by single clicking a value of 0 ms can be used. The default value is 300 ms.

The option ‘zoomcoordinates‘ determines if the coordinates of the zoom box are drawn at the edges while zooming. This is on by default.

The option ‘polardistance‘ determines if the distance to the ruler is also shown in polar coordinates. This corresponds to the default key binding ’5’.

The ‘format‘ option takes a fprintf like format string which determines how floating point numbers are printed to the drivers window and the clipboard. The default is "% #g".

‘clipboardformat‘ and ‘mouseformat‘ are used for formatting the text on Button1 and Button2 actions – copying the coordinates to the clipboard and temporarily annotating the mouse position. This corresponds to the key bindings ’1’, ’2’, ’3’, ’4’ (see the drivers’s help window). If the argument is a string this string is used as c format specifier and should contain two float specifiers, e.g. ‘set mouse mouseformat "mouse = %5.2g, %10.2f"‘. Use ‘set mouse mouseformat ""‘ to turn this string off again.

The following formats are available (format 6 may only be selected if the format string was specified already):

```
 0   real coordinates in  brackets e.g. [1.23, 2.45]
 1   real coordinates w/o brackets e.g.  1.23, 2.45
 2   x == timefmt                       [(as set by Section 2.21.67
[timefmt], page 147), 2.45]
 3   x == date                          [31. 12. 1999, 2.45]
 4   x == time                          [23:59, 2.45]
 5   x == date / time                   [31. 12. 1999 23:59, 2.45]
 6   alt. format, specified as string   ""
```

Choose the option ‘labels‘ to get real gnuplot labels on Button 2. (The default is ‘nolabels‘ which makes Button 2 drawing only temporary annotations at the mouse positions). The labels are drawn with the current setting of ‘mouseformat‘. ‘labeloptions‘ controls which options are passed to the Section 2.21.29 [label], page 104 command. The default is "pointstyle 1" which will plot a small plus at the label position. Note that the pointsize is taken from the Section 2.21.51 [pointsize], page 126 command. Labels can be removed by holding the Ctrl-Key down while clicking with Button 2 on the label’s point. The threshold for how close you must be to the label is also determined by the Section 2.21.51 [pointsize], page 126.

If the option ‘zoomjump‘ is on, the mouse pointer will be automatically offset a small distance after starting a zoom region with button 3. This can be useful to avoid a tiny (or even empty) zoom region. ‘zoomjump‘ is off by default.

If the option ‘verbose‘ is turned on the communication commands are shown during execution. This option can also be toggled by hitting ‘6‘ in the driver’s window. ‘verbose‘ is off by default.

Press ’h’ in the driver’s window for a short summary of the mouse and key bindings. This will also display user defined bindings or ‘hotkeys‘ which can be defined using the Section 1.15.1 [bind], page 26 command, see help for Section 1.15.1 [bind], page 26. Note, that user defined ‘hotkeys‘ may override the default bindings.

Press ’q’ in the driver’s window to close the window. This key cannot be overridden with the Section 1.15.1 [bind], page 26 command.

See also help for Section 1.15.1 [bind], page 26 and Section 2.21.29 [label], page 104.

2.21.37.1 x11_mouse

If multiple X11 plot windows have been opened using the ‘set term x11 <n>‘ terminal option, then only the current plot window supports the entire range of mouse commands and hotkeys. The other windows will, however, continue to display mouse coordinates at the lower left.

This was valid only for gnuplot 4.0 (now mousing is on by default as in the case of other screen terminals): X11 mouse support is turned on by default if standard input comes from a terminal (tty). Mouse support is turned off if standard input does not come from a tty, e.g. a pipe. If you want to use mouse support while writing to gnuplot from a pipe, the mouse must be turned on *before* starting the x11 driver, e.g. immediately after startup with the explicit command ‘set mouse‘. Beware: on some UNIX flavours, special input devices as /dev/null might not be ‘select-able‘; turning on the mouse when using such devices will hang gnuplot.

2.21.38 multiplot

The command Section 2.21.38 [multiplot], page 111 places ‘gnuplot‘ in the multiplot mode, in which several plots are placed on the same page, window, or screen.

Syntax:

```
      set multiplot { layout <rows>,<cols>
                      {rowsfirst|columnsfirst} {downwards|upwards}
                      {scale <xscale>{,<yscale>}} {off-
set <xoff>{,<yoff>}}
                    }
      unset multiplot
```

For some terminals, no plot is displayed until the command Section 2.21.38 [multiplot], page 111 is given, which causes the entire page to be drawn and then returns gnuplot to its normal single-plot mode. For other terminals, each separate ‘plot‘ command produces an updated display, either by redrawing all previous ones and the newly added plot, or by just adding the new plot to the existing display.

The area to be used by the next plot is not erased before doing the new plot. The Section 2.3 [clear], page 36 command can be used to do this if wanted, as is typically the case for "inset" plots.

Any labels or arrows that have been defined will be drawn for each plot according to the current size and origin (unless their coordinates are defined in the ‘screen‘ system). Just about everything else that can be ‘set‘ is applied to each plot, too. If you want something to appear only once on the page, for instance a single time stamp, you'll need to put a ‘set time‘/‘unset time‘ pair around one of the ‘plot‘, ‘splot‘ or Section 2.17 [replot], page 71 commands within the Section 2.21.38 [multiplot], page 111/Section 2.21.38 [multiplot], page 111 block.

The commands Section 2.21.45 [origin], page 115 and Section 2.21.57 [size], page 129 must be used to correctly position each plot if no ‘layout‘ is specified or if fine tuning is desired. See Section 2.21.45 [origin], page 115 and Section 2.21.57 [size], page 129 for details of their usage.

Example:

```
set size 0.7,0.7
set origin 0.1,0.1
set multiplot
set size 0.4,0.4
set origin 0.1,0.1
plot sin(x)
set size 0.2,0.2
set origin 0.5,0.5
plot cos(x)
unset multiplot
```

displays a plot of cos(x) stacked above a plot of sin(x). Note the initial Section 2.21.57 [size], page 129 and Section 2.21.45 [origin], page 115. While these are not always required, their inclusion is recommended. Some terminal drivers require that bounding box information be available before any plots can be made, and the form given above guarantees that the bounding box will include the entire plot array rather than just the bounding box of the first plot.

Section 2.21.57 [size], page 129 and Section 2.21.45 [origin], page 115 refer to the entire plotting area used for each plot. If you want to have the axes themselves line up, you can guarantee that the margins are the same size with the Section 2.21.36 [margin], page 109 commands. See Section 2.21.36 [margin], page 109 for their use. Note that the margin settings are absolute, in character units, so the appearance of the graph in the remaining space will depend on the screen size of the display device, e.g., perhaps quite different on a video display and a printer.

With the ‘layout‘ option you can generate simple multiplots without having to give the Section 2.21.57 [size], page 129 and Section 2.21.45 [origin], page 115 commands before each plot: Those are generated automatically, but can be overridden at any time. With ‘layout‘ the display will be divided by a grid with <rows> rows and <cols> columns. This grid is filled rows first or columns first depending on whether the corresponding option is given in the multiplot command. The stack of plots can grow ‘downwards‘ or ‘upwards‘. Default is ‘rowsfirst‘ and ‘downwards‘.

Each plot can be scaled by ‘scale‘ and shifted with ‘offset‘; if the y-values for scale or offset are omitted, the x-value will be used. Section 2.21.38 [multiplot], page 111 will turn off the automatic layout and restore the values of Section 2.21.57 [size], page 129 and Section 2.21.45 [origin], page 115 as they were before 'set multiplot layout'.

Example:

```
set size 1,1
set origin 0,0
set multiplot layout 3,2 columnsfirst scale 1.1,0.9
[ up to 6 plot commands here ]
unset multiplot
```

The above example will produce 6 plots in 2 columns filled top to bottom, left to right. Each plot will have a horizontal size of 1.1/2 and a vertical size of 0.9/3.

See also multiplot demo (multiplt.dem)

2.21.39 mx2tics

Minor tic marks along the x2 (top) axis are controlled by Section 2.21.39 [mx2tics], page 113. Please see Section 2.21.40 [mxtics], page 113.

2.21.40 mxtics

Minor tic marks along the x axis are controlled by Section 2.21.40 [mxtics], page 113. They can be turned off with Section 2.21.40 [mxtics], page 113. Similar commands control minor tics along the other axes.

Syntax:

```
set mxtics {<freq> | default}
unset mxtics
show mxtics
```

The same syntax applies to Section 2.21.42 [mytics], page 114, Section 2.21.43 [mztics], page 114, Section 2.21.39 [mx2tics], page 113, Section 2.21.41 [my2tics], page 114 and 'mcbtics'.

<freq> is the number of sub-intervals (NOT the number of minor tics) between major tics (the default for a linear axis is either two or five depending on the major tics, so there are one or four minor tics between major tics). Selecting 'default' will return the number of minor ticks to its default value.

If the axis is logarithmic, the number of sub-intervals will be set to a reasonable number by default (based upon the length of a decade). This will be overridden if <freq> is given. However the usual minor tics (2, 3, ..., 8, 9 between 1 and 10, for example) are obtained by setting <freq> to 10, even though there are but nine sub-intervals.

To set minor tics at arbitrary positions, use the ("<label>" <pos> <level>, ...) form of 'set {x|x2|y|y2|z}tics' with <label> empty and <level> set to 1.

The 'set m{x|x2|y|y2|z}tics' commands work only when there are uniformly spaced major tics. If all major tics were placed explicitly by 'set {x|x2|y|y2|z}tics', then minor tic commands are ignored. Implicit major tics and explicit minor tics can be combined using 'set {x|x2|y|y2|z}tics' and 'set {x|x2|y|y2|z}tics add'.

Examples:

```
set xtics 0, 5, 10
set xtics add (7.5)
set mxtics 5
```

Major tics at 0,5,7.5,10, minor tics at 1,2,3,4,6,7,8,9

```
set logscale y
set format y ""
set ytics 1e-6, 10, 1
set ytics add ("1" 1, ".1" 0.1, ".01" 0.01, "10^-3" 0.001, \
               "10^-4" 0.0001)
set mytics 10
```

Major tics with special formatting, minor tics at log positions

By default, minor tics are off for linear axes and on for logarithmic axes. They inherit the settings for ‘axis|border‘ and ‘{no}mirror‘ specified for the major tics. Please see Section 2.21.88 [xtics], page 155 for information about these.

2.21.41 my2tics

Minor tic marks along the y2 (right-hand) axis are controlled by Section 2.21.41 [my2tics], page 114. Please see Section 2.21.40 [mxtics], page 113.

2.21.42 mytics

Minor tic marks along the y axis are controlled by Section 2.21.42 [mytics], page 114. Please see Section 2.21.40 [mxtics], page 113.

2.21.43 mztics

Minor tic marks along the z axis are controlled by Section 2.21.43 [mztics], page 114. Please see Section 2.21.40 [mxtics], page 113.

2.21.44 offsets

Offsets provide a mechanism to put a boundary around the data inside of an autoscaled graph.

Syntax:

```
set offsets <left>, <right>, <top>, <bottom>
unset offsets
show offsets
```

Each offset may be a constant or an expression. Each defaults to 0. Left and right offsets are given in units of the x axis, top and bottom offsets in units of the y axis. A positive offset expands the graph in the specified direction, e.g., a positive bottom offset makes ymin more negative. Negative offsets, while permitted, can have unexpected interactions with autoscaling and clipping.

Offsets are ignored in ‘splot‘s.

Example:

```
set offsets 0, 0, 2, 2
plot sin(x)
```

This graph of sin(x) will have a y range [-3:3] because the function will be autoscaled to [-1:1] and the vertical offsets are each two.

2.21.45 origin

The Section 2.21.45 [origin], page 115 command is used to specify the origin of a plotting surface (i.e., the graph and its margins) on the screen. The coordinates are given in the 'screen' coordinate system (see 'coordinates' for information about this system).

Syntax:

```
set origin <x-origin>,<y-origin>
```

2.21.46 output

By default, screens are displayed to the standard output. The Section 2.21.46 [output], page 115 command redirects the display to the specified file or device.

Syntax:

```
set output {"<filename>"}
show output
```

The filename must be enclosed in quotes. If the filename is omitted, any output file opened by a previous invocation of Section 2.21.46 [output], page 115 will be closed and new output will be sent to STDOUT. (If you give the command 'set output "STDOUT"', your output may be sent to a file named "STDOUT"! ["May be", not "will be", because some terminals, like 'x11', ignore Section 2.21.46 [output], page 115.])

MSDOS users should note that the \ character has special significance in double-quoted strings, so single-quotes should be used for filenames in different directories.

When both 'set terminal' and Section 2.21.46 [output], page 115 are used together, it is safest to give 'set terminal' first, because some terminals set a flag which is needed in some operating systems. This would be the case, for example, if the operating system needs to know whether or not a file is to be formatted in order to open it properly.

On machines with popen functions (Unix), output can be piped through a shell command if the first non-whitespace character of the filename is '|'. For instance,

```
set output "|lpr -Plaser filename"
set output "|lp -dlaser filename"
```

On MSDOS machines, 'set output "PRN"' will direct the output to the default printer. On VMS, output can be sent directly to any spooled device. It is also possible to send the output to DECnet transparent tasks, which allows some flexibility.

2.21.47 parametric

The 'set parametric' command changes the meaning of 'plot' ('splot') from normal functions to parametric functions. The command 'unset parametric' restores the plotting style to normal, single-valued expression plotting.

Syntax:

```
set parametric
unset parametric
show parametric
```

For 2-d plotting, a parametric function is determined by a pair of parametric functions operating on a parameter. An example of a 2-d parametric function would be ‘plot sin(t),cos(t)‘, which draws a circle (if the aspect ratio is set correctly—see Section 2.21.57 [size], page 129). ‘gnuplot‘ will display an error message if both functions are not provided for a parametric ‘plot‘.

For 3-d plotting, the surface is described as x=f(u,v), y=g(u,v), z=h(u,v). Therefore a triplet of functions is required. An example of a 3-d parametric function would be ‘cos(u)*cos(v),cos(u)*sin(v),sin(u)‘, which draws a sphere. ‘gnuplot‘ will display an error message if all three functions are not provided for a parametric ‘splot‘.

The total set of possible plots is a superset of the simple f(x) style plots, since the two functions can describe the x and y values to be computed separately. In fact, plots of the type t,f(t) are equivalent to those produced with f(x) because the x values are computed using the identity function. Similarly, 3-d plots of the type u,v,f(u,v) are equivalent to f(x,y).

Note that the order the parametric functions are specified is xfunction, yfunction (and zfunction) and that each operates over the common parametric domain.

Also, the ‘set parametric‘ function implies a new range of values. Whereas the normal f(x) and f(x,y) style plotting assume an xrange and yrange (and zrange), the parametric mode additionally specifies a trange, urange, and vrange. These ranges may be set directly with Section 2.21.70 [trange], page 149, Section 2.21.71 [urange], page 149, and Section 2.21.75 [vrange], page 150, or by specifying the range on the ‘plot‘ or ‘splot‘ commands. Currently the default range for these parametric variables is [-5:5]. Setting the ranges to something more meaningful is expected.

2.21.48 plot

The ‘show plot‘ command shows the current plotting command as it results from the last ‘plot‘ and/or ‘splot‘ and possible subsequent Section 2.17 [replot], page 71 commands.

In addition, the ‘show plot add2history‘ command adds this current plot command into the ‘history‘. It is useful if you have used Section 2.17 [replot], page 71 to add more curves to the current plot and you want to edit the whole command now.

2.21.49 pm3d

pm3d is an ‘splot‘ style for drawing palette-mapped 3d and 4d data as color/gray maps and surfaces. It uses a pm3d algorithm which allows plotting gridded as well as non-gridded data without preprocessing, even when the data scans do not have the same number of points.

Drawing of color surfaces is available on terminals supporting filled colored polygons with color mapping specified by Section 2.21.50 [palette], page 120. Currently supported terminals include

```
Screen terminals:
  OS/2 Presentation Manager
```

```
  X11
  Linux VGA (vgagl)
  GGI
  Windows
  AquaTerm (Mac OS X)
Files:
  PostScript
  pslatex, pstex, epslatex
  gif, png, jpeg
  (x)fig
  tgif
  cgm
  pdf
  svg
```

Let us first describe how a map/surface is drawn. The input data come from an evaluated function or from an Section 2.21.50.4 [file], page 125. Each surface consists of a sequence of separate scans (isolines). The pm3d algorithm fills the region between two neighbouring points in one scan with another two points in the next scan by a gray (or color) according to z-values (or according to an additional 'color' column, see help for Section 2.12.1.9 [using], page 61) of these 4 corners; by default the 4 corner values are averaged, but this can be changed by the option ‘corners2color‘. In order to get a reasonable surface, the neighbouring scans should not cross and the number of points in the neighbouring scans should not differ too much; of course, the best plot is with scans having same number of points. There are no other requirements (e.g. the data need not be gridded). Another advantage is that the pm3d algorithm does not draw anything outside of the input (measured or calculated) region.

Surface coloring works with the following input data:

1. splot of function or of data file with one or three data columns: The gray/color scale is obtained by mapping the averaged (or ‘corners2color‘) z-coordinate of the four corners of the above-specified quadrangle into the range [min_color_z,max_color_z] of Section 2.21.114 [zrange], page 162 or Section 2.21.118 [cbrange], page 162 providing a gray value in the range [0:1]. This value can be used directly as the gray for gray maps. The normalized gray value can be further mapped into a color—see Section 2.21.50 [palette], page 120 for the complete description.

2. splot of data file with two or four data columns: The gray/color value is obtained by using the last-column coordinate instead of the z-value, thus allowing the color and the z-coordinate be mutually independent. This can be used for 4d data drawing.

Other notes:

1. The term 'scan' referenced above is used more among physicists than the term 'iso_curve' referenced in gnuplot documentation and sources. You measure maps recorded one scan after another scan, that's why.

2. The 'gray' or 'color' scale is a linear mapping of a continuous variable onto a smoothly varying palette of colors. The mapping is shown in a rectangle next to the main plot. This

documentation refers to this as a "colorbox", and refers to the indexing variable as lying on the colorbox axis. See 'set colorbox', Section 2.21.118 [cbrange], page 162.

3. To use pm3d coloring to generate a two-dimensional plot rather than a 3D surface, use 'set view map' or 'set pm3d map'.

Syntax (the options can be given in any order):

```
          set pm3d {
                     { at <bst combination> }
                     { interpolate <steps in scan>,<steps between scans> }
                     { scansautomatic | scansforward | scansbackward }
                     { flush { begin | center | end } }
                     { ftriangles | noftriangles }
                     { clip1in | clip4in }
                     { cor-
    ners2color { mean|geomean|median|min|max|c1|c2|c3|c4 } }
                     { hidden3d <linestyle> | nohidden3d }
                     { implicit | explicit }
                     { map }
                   }
          show pm3d
          unset pm3d
```

Color surface is drawn if data or function Section 2.21.58 [style], page 130 is set to pm3d globally or via 'with' option, or if the option 'implicit' is on—then the pm3d surface is combined with the line surface mesh. See bottom of this section for mode details.

Color surface can be drawn at the base or top (then it is a gray/color planar map) or at z-coordinates of surface points (gray/color surface). This is defined by the 'at' option with a string of up to 6 combinations of 'b', 't' and 's'. For instance, 'at b' plots at bottom only, 'at st' plots firstly surface and then top map, while 'at bstbst' will never by seriously used.

Colored quadrangles are plotted one after another. When plotting surfaces ('at s'), the later quadrangles overlap (overdraw) the previous ones. (Gnuplot is not virtual reality tool to calculate intersections of filled polygon meshes.) You may try to switch between 'scansforward' and 'scansbackward' to force the first scan of the data to be plotted first or last. The default is 'scansautomatic' where gnuplot makes a guess about scans order.

If two subsequent scans do not have same number of points, then it has to be decided whether to start taking points for quadrangles from the beginning of both scans ('flush begin'), from their ends ('flush end') or to center them ('flush center'). Note, that 'flush (center|end)' are incompatible with 'scansautomatic': if you specify 'flush center' or 'flush end' and 'scansautomatic' is set, it is silently switched to 'scansforward'.

If two subsequent scans do not have the same number of points, the option 'ftriangles' specifies whether color triangles are drawn at the scan tail(s) where there are not enough points in either of the scan. This can be used to draw a smooth map boundary.

Clipping with respect to x, y coordinates of quadrangles can be done in two ways. 'clip1in': all 4 points of each quadrangle must be defined and at least 1 point of the quadrangle must lie in the x and y ranges. 'clip4in': all 4 points of each quadrangle must lie in the x and y ranges.

There is a single gray/color value associated to each drawn pm3d quadrangle (no smooth color change among vertices). The value is calculated from z-coordinates from the surrounding corners according to ‘corners2color <option>‘. The options ’mean’ (default), ’geomean’ and ’median’ produce various kinds of surface color smoothing, while options ’min’ and ’max’ choose minimal or maximal value, respectively. This may not be desired for pixel images or for maps with sharp and intense peaks, in which case the options ’c1’, ’c2’, ’c3’ or ’c4’ can be used instead to assign the quadrangle color based on the z-coordinate of only one corner. Some experimentation may be needed to determine which corner corresponds to ’c1’, as the orientation depends on the drawing direction. Because the pm3d algorithm does not extend the colored surface outside the range of the input data points, the ’c<j>’ coloring options will result in pixels along two edges of the grid not contributing to the color of any quadrangle. For example, applying the pm3d algorithm to the 4x4 grid of data points in script ‘demo/pm3d.dem‘ (please have a look) produces only (4-1)x(4-1)=9 colored rectangles.

Another drawing algorithm, which would draw quadrangles around a given node by taking corners from averaged (x,y)-coordinates of its surrounding 4 nodes while using node’s color, could be implemented in the future. This is already done for drawing images (2D grids) via ‘image‘ and ‘rgbimage‘ styles.

Notice that ranges of z-values and color-values for surfaces are adjustable independently by Section 2.21.114 [zrange], page 162, Section 2.21.118 [cbrange], page 162, as well as ‘set log‘ for z or cb. Maps can be adjusted by the cb-axis only; see also ‘set view map‘ and ‘set colorbox‘.

The option Section 2.21.25 [hidden3d], page 97 takes as the argument a linestyle which must be created by ‘set style line ...‘. (The style need not to be present when setting pm3d, but it must be present when plotting). If set, lines are drawn using the specified line style, taking into account hidden line removal. This is by far more efficient than using the command Section 2.21.25 [hidden3d], page 97 as it doesn’t really calculate hidden line removal, but just draws the filled polygons in the correct order. So the recommended choice when using pm3d is

```
set pm3d at s hidden3d 100
set style line 100 lt 5 lw 0.5
unset hidden3d
unset surf
splot x*x+y*y
```

There used to be an option {transparent|solid} to this command. Now you get the same effect from ‘set grid {front|layerdefault}‘, respectively.

The ‘set pm3d map‘ is an abbreviation for ‘set pm3d at b‘; ‘set view map‘; pm3d; pm3d;. It is used for backwards compatibility, when ‘set view map‘ was not available. Take care that you properly use Section 2.21.114 [zrange], page 162 and Section 2.21.118 [cbrange], page 162 for input data point filtering and color range scaling, respectively; and also ‘set (no)surface‘ seems to have a (side?) effect.

The option ‘interpolate‘ will interpolate grid points into a finer mesh, and color each quadrangle appropriately. For data files, this will smoothen the color surface, and enhance spikes in a color surface. For functions, interpolation makes little sense, except to trade off precision for memory. It would usually make more sense to use Section 2.21.56 [samples], page 128 and Section 2.21.27 [isosamples], page 100 when working with functions.

The coloring setup as well as the color box drawing are determined by Section 2.21.50 [palette], page 120. There can be only one palette for the current plot. Drawing of several surfaces with different palettes can be achieved by Section 2.21.38 [multiplot], page 111 with fixed Section 2.21.45 [origin], page 115 and Section 2.21.57 [size], page 129; don't forget to use 'set palette maxcolors' when your terminal is running out of available colors.

On gnuplot start-up, mode is 'explicit'. For historical and thus compatibility reasons, commands 'set pm3d;' (i.e. no options) and 'set pm3d at X ...' (i.e. 'at' is the first option) sets mode 'implicit'. Further, 'set pm3d;' sets up the other options to their default.

If the option 'implicit' is on, all surface plots will be plotted additionally to the default type, e.g.

```
splot 'fred.dat' with lines, 'lola.dat' with lines
```

would give both plots (meshes) additionally to a pm3d surface. That's what you are used to after 'set pm3d;'.

If the option 'explicit' is on (or 'implicit' is off) only plots specified by the pm3d attribute are plotted with a pm3d surface, e.g.:

```
splot 'fred.dat' with lines, 'lola.dat' with pm3d
```

would plot 'fred.dat' with lines (and only lines) and 'lola.dat' with a pm3d surface.

If you set the default data or function style to pm3d, e.g.:

```
set style data pm3d
```

then the options 'implicit' and 'explicit' have no effect.

Note that when plotting several plots, they are plotted in the order given on the command line. This can be of interest especially for filled surfaces which can overwrite and therefore hide part of earlier plots.

If pm3d is specified in the 'splot' command line, then it accepts the 'at' option. The following plots draw three color surfaces at different altitudes:

```
set border 4095
set pm3d at s
splot 10*x with pm3d at b, x*x-y*y, x*x+y*y with pm3d at t
```

See also help for Section 2.21.50 [palette], page 120, Section 2.21.118 [cbrange], page 162, 'set colorbox', pm3d and definitely the demo file 'demo/pm3d.dem'.

2.21.50 palette

Palette is a color storage for use by pm3d, filled color contours or polygons, color histograms, color gradient background, and whatever it is or it will be implemented... Here it stands for a palette of smooth "continuous" colors or grays, but let's call it just a palette.

Color palettes require terminal entries for filled color polygons and palettes of smooth colors, are currently available for terminals listed in help for pm3d. The range of color values are adjustable independently by Section 2.21.118 [cbrange], page 162 and 'set log cb'. The whole color palette is visualized in the 'colorbox'.

Syntax:

```
        set palette
        set palette {
                   { gray | color }
                   { gamma <gamma> }
                   {   rgbformulae <r>,<g>,<b>
                     | defined { ( <gray1> <color1> {, <grayN> <col-
orN>}... ) }
                     | file '<filename>' {datafile-modifiers}
                     | functions <R>,<G>,<B>
                   }
                   { model { RGB | HSV | CMY | YIQ | XYZ } }
                   { positive | negative }
                   { nops_allcF | ps_allcF }
                   { maxcolors <maxcolors> }
                 }
        show palette
        show palette palette <n> {{float | int}}
        show palette gradient
        show palette fit2rgbformulae
        show palette rgbformulae
        show palette colornames
```

Section 2.21.50 [palette], page 120 (i.e. without options) sets up the default values. Otherwise, the options can be given in any order. Section 2.21.50 [palette], page 120 shows the current palette properties.

'show palette gradient' displays the gradient defining the palette (if appropriate). Section 2.21.50.1 [rgbformulae], page 122 prints the available fixed gray –> color transformation formulae. Section 2.21.50.7 [colornames], page 126 prints the implemented color names.

'show palette palette <n>' prints to screen or to the file given by Section 2.21.46 [output], page 115 table of RGB triplets calculated for the current palette settings and a palette having <n> discrete colors. The default wide table can be limited to 3 columns of r,g,b float values [0..1] or integer values [0..255] by options float or int, respectively. This way, the current gnuplot color palette can be loaded into other imaging applications, for example Octave. Additionally to this textual list of RGB table, you can enjoy command Section 2.21.50 [palette], page 120 to draw graphically the R,G,B profiles for the current palette.

The following options determine the coloring properties.

Figure using this palette can be 'gray' or 'color'. For instance, in pm3d color surfaces the gray of each small spot is obtained by mapping the averaged z-coordinate of the 4 corners of surface quadrangles into the range [min_z,max_z] providing range of grays [0:1]. This value can be used directly as the gray for gray maps. The color map requires a transformation gray –> (R,G,B), i.e. a mapping [0:1] –> ([0:1],[0:1],[0:1]).

Basically two different types of mappings can be used: Analytic formulae to convert gray to color, or discrete mapping tables which are interpolated. Section 2.21.50.1 [rgbformulae], page 122 and 'palette functions' use analytic formulae whereas 'palette defined' and Section 2.21.50.4 [file], page 125 use interpolated tables. Section 2.21.50.1 [rgbformulae], page 122 reduces the size of postscript output to a minimum.

The command 'show palette fit2rgbformulae' finds the best matching Section 2.21.50.1 [rgbformulae], page 122 for the current Section 2.21.50 [palette], page 120. Naturally, it makes sense to use it for non-rgbformulae palettes. This command can be found useful mainly for external programs using the same rgbformulae definition of palettes as gnuplot, like zimg (http://zimg.sourceforge.net

```
).
```

'set palette gray' switches to a gray only palette. Section 2.21.50.1 [rgbformulae], page 122, 'set palette defined', Section 2.21.50.4 [file], page 125 and 'set palette functions' switch to a color mapping. 'set palette color' is an easy way to switch back from the gray palette to the last color mapping.

Automatic gamma correction via 'set palette gamma <gamma>' can be done for gray maps only ('set palette gray'). Linear mapping to gray is for gamma equals 1, see Section 2.21.50 [palette], page 120. Gamma is ignored for color mappings.

Most terminals support only discrete number of colors (e.g. 256 colors in gif). All entries of the palette remaining after the default gnuplot linetype colors declaration are allocated for pm3d by default. Then Section 2.21.38 [multiplot], page 111 could fail if there are no more color positions in the terminal available. Then you should use 'set palette maxcolors <maxcolors>' with a reasonably small value. This option can also be used to separate levels of z=constant in discrete steps, thus to emulate filled contours. Default value of 0 stays for allocating all remaining entries in the terminal palette or for to use exact mapping to RGB.

RGB color space might not be the most useful color space to work in. For that reason you may change the color space with 'model' to one of 'RGB', 'HSV', 'CMY', 'YIQ' and 'XYZ'. Using color names for 'set palette defined' tables and a color space other than RGB will result in funny colors. All explanation have been written for RGB color space, so please note, that 'R' can be 'H', 'C', 'Y', or 'X', depending on the actual color space ('G' and 'B' accordingly).

All values for all color spaces are limited to [0,1].

RGB stands for Red, Green and Blue; CMY stands for Cyan, Magenta and Yellow; HSV stands for Hue, Saturation, and Value; YIQ is the color model used by the U.S. Commercial Color Television Broadcasting, it is basically an RGB recoding with downward compatibility for black and white television; XYZ are the three primary colors of the color model defined by the 'Commission Internationale de l'Eclairage' (CIE). For more information on color models see: http://www.cs.rit.edu/~ncs/color/glossary.htm and http://cs.fit.edu/wds/classes/cse5255/cse5255/davis/index.html

2.21.50.1 rgbformulae

For Section 2.21.50.1 [rgbformulae], page 122 three suitable mapping functions have to be chosen. This is done via 'rgbformulae <r>,<g>,<b>'. The available mapping functions are listed by Section 2.21.50.1 [rgbformulae], page 122. Default is '7,5,15', some other examples are '3,11,6', '21,23,3' or '3,23,21'. Negative numbers, like '3,-11,-6', mean inverted color (i.e. 1-gray passed into the formula, see also 'positive' and 'negative' options below).

Some nice schemes in RGB color space

```
7,5,15   ... traditional pm3d (black-blue-red-yellow)
3,11,6   ... green-red-violet
```

```
   23,28,3  ... ocean (green-blue-
white); try also all other permutations
   21,22,23 ... hot (black-red-yellow-white)
   30,31,32 ... color printable on gray (black-blue-violet-yellow-
white)
   33,13,10 ... rainbow (blue-green-yellow-red)
   34,35,36 ... AFM hot (black-red-yellow-white)
```

A full color palette in HSV color space

```
   3,2,2    ... red-yellow-green-cyan-blue-magenta-red
```

Please note that even if called Section 2.21.50.1 [rgbformulae], page 122 the formulas might actually determine the <H>,<S>,<V> or <X>,<Y>,<Z> or ... color components as usual.

Use 'positive' and 'negative' to invert the figure colors.

Note that it is possible to find a set of the best matching rgbformulae for any other color scheme by the command

```
   show palette fit2rgbformulae
```

2.21.50.2 defined

Gray-to-rgb mapping can be manually set by use of 'palette defined': A color gradient is defined and used to give the rgb values. Such a gradient is a piecewise linear mapping from gray values in [0,1] to the RGB space [0,1]x[0,1]x[0,1]. You have to specify the gray values and the corresponding RGB values in between a linear interpolation shall take place:

Syntax:

```
      set palette  defined { ( <gray1> <color1> {, <grayN> <col-
orN>}... ) }
```

<grayX> are gray values which are mapped to [0,1] and <colorX> are the corresponding rgb colors. The color can be specified in three different ways:

```
     <color> :=  { <r> <g> <b> | '<color-name>' | '#rrggbb' }
```

Either by three numbers (each in [0,1]) for red, green and blue, separated by whitespace, or the name of the color in quotes or X style color specifiers also in quotes. You may freely mix the three types in a gradient definition, but the named color "red" will be something strange if RGB is not selected as color space. Use Section 2.21.50.7 [colornames], page 126 for a list of known color names.

Please note, that even if written as <r>, this might actually be the <H> component in HSV color space or <X> in CIE-XYZ space, or ... depending on the selected color model.

The <gray> values have to form an ascending sequence of real numbers; the sequence will be automatically rescaled to [0,1].

'set palette defined' (without a gradient definition in braces) switches to RGB color space and uses a preset full-spectrum color gradient. Use 'show palette gradient' to display the gradient.

Examples:

To produce a gray palette (useless but instructive) use:

```
set palette model RGB
set palette defined ( 0 "black", 1 "white" )
```

To produce a blue yellow red palette use (all equivalent):

```
set palette defined ( 0 "blue", 1 "yellow", 2 "red" )
set palette defined ( 0 0 0 1, 1 1 1 0, 2 1 0 0 )
set palette defined ( 0 "#0000ff", 1 "#ffff00", 2 "ff0000" )
```

To produce some rainbow-like palette use:

```
set palette defined ( 0 "blue", 3 "green", 6 "yel-
low", 10 "red" )
```

Full color spectrum within HSV color space:

```
set palette model HSV
set palette defined ( 0 0 1 1, 1 1 1 1 )
set palette de-
fined ( 0 0 1 0, 1 0 1 1, 6 0.8333 1 1, 7 0.8333 0 1)
```

To produce a palette with few colors only use:

```
set palette model RGB maxcolors 4
set palette defined ( 0 "blue", 1 "green", 2 "yellow", 3 "red" )
```

'Traffic light' palette (non-smooth color jumps at gray = 1/3 and 2/3).

```
set palette model RGB
set palette defined (0 "dark-green", 1 "green", 1 "yellow", \
                     2 "dark-yellow", 2 "red", 3 "dark-red" )
```

2.21.50.3 functions

Use 'set palette functions <Rexpr>, <Gexpr>, <Bexpr>' to define three formulae for the R(gray), G(gray) and B(gray) mapping. The three formulae may depend on the variable 'gray' which will take values in [0,1] and should also produce values in [0,1]. Please note that <Rexpr> might be a formula for the H-value if HSV color space has been chosen (same for all other formulae and color spaces).

Examples:

To produce a full color palette use:

```
set palette model HSV functions gray, 1, 1
```

A nice black to gold palette:

```
set palette model XYZ functions gray**0.35, gray**0.5, gray**0.8
```

A gamma-corrected black and white palette

```
      gamma = 2.2
      color(gray) = gray**(1./gamma)
      set palette model RGB func-
tions color(gray), color(gray), color(gray)
```

2.21.50.4 file

Section 2.21.50.4 [file], page 125 is basically a 'set palette defined (<gradient>)' where <gradient> is read from a datafile. Either 4 columns (gray,R,G,B) or just three columns (R,G,B) have to be selected via the Section 2.12.1.9 [using], page 61 data file modifier. In the three column case, the line number will be used as gray. The gray range is automatically rescaled to [0,1]. The file is read as a normal data file, so all datafile modifiers can be used. Please note, that 'R' might actually be e.g. 'H' if HSV color space is selected.

As usual <filename> may be ''-'' which means that the data follow the command inline and are terminated by a single 'e' on a line of its own.

Use 'show palette gradient' to display the gradient.

Examples:

Read in a palette of RGB triples each in range [0,255]:

```
set palette file 'some-palette' using ($1/255):($2/255):($3/255)
```

Equidistant rainbow (blue-green-yellow-red) palette:

```
set palette model RGB file "-"
0 0 1
0 1 0
1 1 0
1 0 0
e
```

2.21.50.5 gamma-correction

For gray mappings gamma correction can be turned on by 'set palette gamma <gamma>'. <gamma> defaults to 1.5 which is quite suitable for most terminals.

For color mappings no automatic gamma correction is done by gnuplot. But you may easily implement gamma correction, here an example for a gray scale image by use of explicit functions for the red, green and blue component with slightly different values of gamma

Example:

```
set palette model RGB
set palette functions gray**0.64, gray**0.67, gray**0.70
```

To use gamma correction with interpolated gradients specify intermediate gray values with appropriate colors. Instead of

```
set palette defined ( 0 0 0 0, 1 1 1 1 )
```

use e.g.

```
set palette defined ( 0 0 0 0, 0.5 .73 .73 .73, 1 1 1 1 )
```

or even more intermediate points until the linear interpolation fits the "gamma corrected" interpolation well enough.

2.21.50.6 postscript

In order to reduce the size of postscript files, the gray value and not all three calculated r,g,b values are written to the file. Therefore the analytical formulae are coded directly in the postscript language as a header just before the pm3d drawing, see /g and /cF definitions. Usually, it makes sense to write therein definitions of only the 3 formulae used. But for multiplot or any other reason you may want to manually edit the transformations directly in the postscript file. This is the default option 'nops_allcF'. Using the option 'ps_allcF' writes postscript definitions of all formulae. This you may find interesting if you want to edit the postscript file in order to have different palettes for different surfaces in one graph. Well, you can achieve this functionality by Section 2.21.38 [multiplot], page 111 with fixed Section 2.21.45 [origin], page 115 and Section 2.21.57 [size], page 129.

If pm3d map has been plotted from gridded or almost regular data with an output to a postscript file, then it is possible to reduce the size of this postscript file up to at about 50% by the enclosed awk script 'pm3dCompress.awk'. This you may find interesting if you intend to keep the file for including it into your publication or before downloading a very large file into a slow printer. Usage:

```
awk -f pm3dCompress.awk thefile.ps >smallerfile.ps
```

If pm3d map has been plotted from rectangular gridded data with an output to a postscript file, then it is possible to reduce the file size even more by the enclosed awk script 'pm3dConvertToImage.awk'. Usage:

```
awk -f pm3dConvertToImage.awk <thefile.ps >smallerfile.ps
```

You may manually change the postscript output from gray to color and vice versa and change the definition of <maxcolors>.

2.21.50.7 colornames

Gnuplot knows a limited number of color names. You can use these to define the color range spanned by a pm3d palette, or to assign a terminal-independent colot to a particular linetype or linestyle. To see the list of know color names, use the command Section 2.21.50.7 [colornames], page 126. See Section 2.21.50 [palette], page 120, 'linestyle'.

2.21.51 pointsize

The Section 2.21.51 [pointsize], page 126 command scales the size of the points used in plots.

Syntax:

```
set pointsize <multiplier>
show pointsize
```

The default is a multiplier of 1.0. Larger pointsizes may be useful to make points more visible in bitmapped graphics.

The pointsize of a single plot may be changed on the ‘plot‘ command. See Section 2.12.7 [with], page 68 for details.

Please note that the pointsize setting is not supported by all terminal types.

2.21.52 polar

The ‘set polar‘ command changes the meaning of the plot from rectangular coordinates to polar coordinates.

Syntax:

```
set polar
unset polar
show polar
```

There have been changes made to polar mode in version 3.7, so that scripts for ‘gnuplot‘ versions 3.5 and earlier will require modification. The main change is that the dummy variable t is used for the angle so that the x and y ranges can be controlled independently. Other changes are: 1) tics are no longer put along the zero axes automatically —use ‘set xtics axis nomirror‘; ‘set ytics axis nomirror‘; 2) the grid, if selected, is not automatically polar —use ‘set grid polar‘; 3) the grid is not labelled with angles —use Section 2.21.29 [label], page 104 as necessary.

In polar coordinates, the dummy variable (t) is an angle. The default range of t is [0:2*pi], or, if degree units have been selected, to [0:360] (see Section 2.21.1 [angles], page 74).

The command ‘unset polar‘ changes the meaning of the plot back to the default rectangular coordinate system.

The ‘set polar‘ command is not supported for ‘splot‘s. See the Section 2.21.35 [mapping], page 108 command for similar functionality for ‘splot‘s.

While in polar coordinates the meaning of an expression in t is really r = f(t), where t is an angle of rotation. The trange controls the domain (the angle) of the function, and the x and y ranges control the range of the graph in the x and y directions. Each of these ranges, as well as the rrange, may be autoscaled or set explicitly. See Section 2.21.87 [xrange], page 154 for details of all the Section 2.12.5 [ranges], page 66 commands.

Example:

```
set polar
plot t*sin(t)
plot [-2*pi:2*pi] [-3:3] [-3:3] t*sin(t)
```

The first ‘plot‘ uses the default polar angular domain of 0 to 2*pi. The radius and the size of the graph are scaled automatically. The second ‘plot‘ expands the domain, and restricts the size of the graph to [-3:3] in both directions.

You may want to 'set size square' to have 'gnuplot' try to make the aspect ratio equal to unity, so that circles look circular. See also polar demos (polar.dem) and polar data plot (poldat.dem).

2.21.53 print

The 'set print' command redirects the output of the 'print' command to a file.

Syntax:

```
set print
set print "-"
set print "<filename>"
set print "<filename>" append
set print "|<shell_command>"
```

Without "<filename>", the output file is restored to <STDERR>. The <filename> "-" means <STDOUT>. The 'append' flag causes the file to be opened in append mode. A <filename> starting with "|" is opened as a pipe to the <shell_command> on platforms that support piping.

2.21.54 rmargin

The command Section 2.21.54 [rmargin], page 128 sets the size of the right margin. Please see Section 2.21.36 [margin], page 109 for details.

2.21.55 rrange

The Section 2.21.55 [rrange], page 128 command sets the range of the radial coordinate for a graph in polar mode. Please see Section 2.21.87 [xrange], page 154 for details.

2.21.56 samples

The sampling rate of functions, or for interpolating data, may be changed by the Section 2.21.56 [samples], page 128 command.

Syntax:

```
set samples <samples_1> {,<samples_2>}
show samples
```

By default, sampling is set to 100 points. A higher sampling rate will produce more accurate plots, but will take longer. This parameter has no effect on data file plotting unless one of the interpolation/approximation options is used. See Section 2.12.1.6 [smooth], page 57 re 2-d data and Section 2.21.10 [cntrparam], page 83 and Section 2.21.16 [dgrid3d], page 90 re 3-d data.

When a 2-d graph is being done, only the value of <samples_1> is relevant.

When a surface plot is being done without the removal of hidden lines, the value of samples specifies the number of samples that are to be evaluated for the isolines. Each iso-v line will have <sample_1> samples and each iso-u line will have <sample_2> samples. If you only specify <samples_1>, <samples_2> will be set to the same value as <samples_1>. See also Section 2.21.27 [isosamples], page 100.

2.21.57 size

The Section 2.21.57 [size], page 129 command scales the displayed size of the plot.

Syntax:

```
set size {{no}square | ratio <r> | noratio} {<xscale>,<yscale>}
show size
```

The <xscale> and <yscale> values are the scaling factors for the size of the plot, which includes the graph and the margins.

'ratio' causes 'gnuplot' to try to create a graph with an aspect ratio of <r> (the ratio of the y-axis length to the x-axis length) within the portion of the plot specified by <xscale> and <yscale>.

The meaning of a negative value for <r> is different. If <r>=-1, gnuplot tries to set the scales so that the unit has the same length on both the x and y axes (suitable for geographical data, for instance). If <r>=-2, the unit on y has twice the length of the unit on x, and so on.

The success of 'gnuplot' in producing the requested aspect ratio depends on the terminal selected. The graph area will be the largest rectangle of aspect ratio <r> that will fit into the specified portion of the output (leaving adequate margins, of course).

'square' is a synonym for 'ratio 1'.

Both 'noratio' and 'nosquare' return the graph to the default aspect ratio of the terminal, but do not return <xscale> or <yscale> to their default values (1.0).

'ratio' and 'square' have no effect on 3-d plots.

Section 2.21.57 [size], page 129 is relative to the default size, which differs from terminal to terminal. Since 'gnuplot' fills as much of the available plotting area as possible by default, it is safer to use Section 2.21.57 [size], page 129 to decrease the size of a plot than to increase it. See 'set terminal' for the default sizes.

On some terminals, changing the size of the plot will result in text being misplaced.

Examples:

To set the size to normal size use:

```
set size 1,1
```

To make the graph half size and square use:

```
set size square 0.5,0.5
```

To make the graph twice as high as wide use:

```
set size ratio 2
```

See also airfoil demo.

2.21.58 style

Default plotting styles are chosen with the 'set style data' and 'set style function' commands. See Section 2.12.7 [with], page 68 for information about how to override the default plotting style for individual functions and data sets. See 'plotting styles' for a complete list of styles.

Syntax:

```
set style function <style>
set style data <style>
show style function
show style data
```

Default styles for specific plotting elements may also be set.

Syntax:

```
set style arrow <n> <arrowstyle>
set style fill <fillstyle>
set style histogram <histogram style options>
set style line <n> <linestyle>
```

2.21.58.1 set style arrow

Each terminal has a default set of arrow and point types, which can be seen by using the command Section 2.25 [test], page 168. Section 2.21.2 [arrow], page 75 defines a set of arrow types and widths and point types and sizes so that you can refer to them later by an index instead of repeating all the information at each invocation.

Syntax:

```
set style arrow <index> default
set style arrow <index> {nohead | head | heads}
                        {size <length>,<angle>{,<backangle>}}
                        {filled | empty | nofilled}
                        {front | back}
                        { {linestyle | ls <line_style>}
                          | {linetype | lt <line_type>}
                            {linewidth | lw <line_width} }
unset style arrow
show style arrow
```

<index> is an integer that identifies the arrowstyle.

If 'default' is given all arrow style parameters are set to their default values.

If the linestyle <index> already exists, only the given parameters are changed while all others are preserved. If not, all undefined values are set to the default values.

Specifying 'nohead' produces arrows drawn without a head—a line segment. This gives you yet another way to draw a line segment on the plot. By default, arrows have one head. Specifying 'heads' draws arrow heads on both ends of the line.

Head size can be controlled by 'size <length>,<angle>' or 'size <length>,<angle>,<backangle>', where '<length>' defines length of each branch of the arrow head and ‘<angle>‘ the angle (in degrees) they make with the arrow. ‘<Length>‘ is in x-axis units; this can be changed by ‘first‘, ‘second‘, ‘graph‘, ‘screen‘, or ‘character‘ before the <length>; see ‘coordinates‘ for details. ‘<Backangle>‘ only takes effect when ‘filled‘ or ‘empty‘ is also used. Then, ‘<backangle>‘ is the angle (in degrees) the back branches make with the arrow (in the same direction as ‘<angle>‘). The ‘fig‘ terminal has a restricted backangle function. It supports three different angles. There are two thresholds: Below 70 degrees, the arrow head gets an indented back angle. Above 110 degrees, the arrow head has an acute back angle. Between these thresholds, the back line is straight.

Specifying ‘filled‘ produces filled arrow heads (if heads are used). Filling is supported on filled-polygon capable terminals, see help of pm3d for their list, otherwise the arrow heads are closed but not filled. The same result (closed but not filled arrow head) is reached by specifying ‘empty‘. Further, filling and outline is obviously not supported on terminals drawing arrows by their own specific routines, like ‘metafont‘, ‘metapost‘, ‘latex‘ or ‘tgif‘.

The line style may be selected from a user-defined list of line styles (see ‘set style line‘) or may be defined here by providing values for ‘<line_type>‘ (an index from the default list of styles) and/or ‘<line_width>‘ (which is a multiplier for the default width).

Note, however, that if a user-defined line style has been selected, its properties (type and width) cannot be altered merely by issuing another Section 2.21.2 [arrow], page 75 command with the appropriate index and ‘lt‘ or ‘lw‘.

If ‘front‘ is given, the arrows are written on top of the graphed data. If ‘back‘ is given (the default), the arrow is written underneath the graphed data. Using ‘front‘ will prevent a arrow from being obscured by dense data.

Examples:

To draw an arrow without an arrow head and double width, use:

```
set style arrow 1 nohead lw 2
set arrow arrowstyle 1
```

```
See also Section 2.21.2 [arrow], page 75 for further examples.
```

2.21.58.2 set style data

The ‘set style data‘ command changes the default plotting style for data plots.

Syntax:

```
set style data <plotting-style>
show style data
```

See ‘plotting styles‘ for the choices. If no choice is given, the choices are listed. ‘show style data‘ shows the current default data plotting style.

2.21.58.3 set style fill

The ‘set style fill‘ command is used to set the style of boxes, histograms, candlesticks and filledcurves.

Syntax:

```
set style fill {empty | solid {<density>} | pattern {<n>}}
               {border {<linetype>} | noborder}
```

The default fillstyle is ‘empty‘.

The ‘solid‘ option causes filling with a solid color, if the terminal supports that. The <density> parameter specifies the intensity of the fill color. At a <density> of 0.0, the box is empty, at <density> of 1.0, the inner area is of the same color as the current linetype. Some terminal types can vary the density continuously; others implement only a few levels of partial fill. If no <density> parameter is given, it defaults to 1.

The ‘pattern‘ option causes filling to be done with a fill pattern supplied by the terminal driver. The kind and number of available fill patterns depend on the terminal driver. If multiple datasets using filled boxes are plotted, the pattern cycles through all available pattern types, starting from pattern <n>, much as the line type cycles for multiple line plots.

The ‘empty‘ option causes filled boxes not to be filled. This is the default. It is equivalent to the ‘solid‘ option with a <density> parameter of zero.

By default, Section 2.21.6 [border], page 79, the box is bounded by a solid line of the current linetype. ‘border <lt>‘ specifies that a border is to be drawn using linetype <lt>. ‘noborder‘ specifies that no bounding lines are drawn.

2.21.58.4 set style function

The ‘set style function‘ command changes the default plotting style for function plots.

Syntax:

```
set style function <plotting-style>
show style function
```

See ‘plotting styles‘ for the choices. If no choice is given, the choices are listed. ‘show style function‘ shows the current default function plotting style.

2.21.58.5 set style line

Each terminal has a default set of line and point types, which can be seen by using the command Section 2.25 [test], page 168. ‘set style line‘ defines a set of line types and widths and point types and sizes so that you can refer to them later by an index instead of repeating all the information at each invocation.

Syntax:

```
      set style line <index> default
      set style line <index> {{line-
type  | lt} <line_type> | <colorspec>}
                             {{linecolor | lc} <colorspec>}
                             {{linewidth | lw} <line_width>}
                             {{pointtype | pt} <point_type>}
                             {{pointsize | ps} <point_size>}
                             {palette}
      unset style line
```

```
show style line
```

If 'default' is given all line style parameters are set to their default values.

If the linestyle <index> already exists, only the given parameters are changed while all others are preserved. If not, all undefined values are set to the default values.

The line and point types are taken from the default types for the terminal currently in use. The line width and point size are multipliers for the default width and size (but note that <point_size> here is unaffected by the multiplier given on Section 2.21.51 [pointsize], page 126).

The defaults for the line and point types is the index. The defaults for the width and size are both unity.

Linestyles created by this mechanism do not replace the default styles; both may be used.

Not all terminals support the 'linewidth' and Section 2.21.51 [pointsize], page 126 features; if not supported, the option will be ignored.

Terminal-independent colors may be assigned using either 'linecolor <colorspec>' or 'linetype <colorspec>', abbreviated 'lc' or 'lt'. This requires giving a RGB color triple, a known palette color name, a fractional index into the current palette, or a constant value from the current mapping of the palette onto cbrange. See 'colors', Section 1.14.1 [colorspec], page 25, Section 2.21.50 [palette], page 120, Section 2.21.50.7 [colornames], page 126, Section 2.21.118 [cbrange], page 162.

'set style line <n> linetype <lt>' will set both a terminal-dependent dot/dash pattern and color. The commands'set style line <n> linecolor <colorspec>' or 'set style line <n> linetype <colorspec>' will set a new line color while leaving the existing dot-dash pattern unchanged.

In 3d mode ('splot' command), the special keyword Section 2.21.50 [palette], page 120 is allowed as a shorthand for "linetype palette z". The color value corresponds to the z-value (elevation) of the splot, and varies smoothly along a line or surface.

Examples: Suppose that the default lines for indices 1, 2, and 3 are red, green, and blue, respectively, and the default point shapes for the same indices are a square, a cross, and a triangle, respectively. Then

```
set style line 1 lt 2 lw 2 pt 3 ps 0.5
```

defines a new linestyle that is green and twice the default width and a new pointstyle that is a half-sized triangle. The commands

```
set style function lines
plot f(x) lt 3, g(x) ls 1
```

will create a plot of f(x) using the default blue line and a plot of g(x) using the user-defined wide green line. Similarly the commands

```
set style function linespoints
plot p(x) lt 1 pt 3, q(x) ls 1
```

will create a plot of p(x) using the default triangles connected by a red line and q(x) using small triangles connected by a green line.

```
splot sin(sqrt(x*x+y*y))/sqrt(x*x+y*y) w l pal
```

creates a surface plot using smooth colors according to Section 2.21.50 [palette], page 120. Note, that this works only on some terminals. See also Section 2.21.50 [palette], page 120, pm3d.

```
set style line 10 linetype 1 linecolor rgb "cyan"
```

will assign linestyle 10 to be a solid cyan line on any terminal that supports rgb colors.

2.21.58.6 plotting styles

The commands 'set style data' and 'set style function' change the default plotting style for subsequent 'plot' and 'splot' commands.

The types used for all line and point styles (i.e., solid, dash-dot, color, etc. for lines; circles, squares, crosses, etc. for points) will be either those specified on the 'plot' or 'splot' command or will be chosen sequentially from the types available to the terminal in use. Use the command Section 2.25 [test], page 168 to see what is available.

None of the styles requiring more than two columns of information (e.g., Section 2.12.2 [errorbars], page 64 or Section 2.12.3 [errorlines], page 64) can be used with 'splot's or function 'plot's. Neither 'boxes', 'filledcurves' nor any of the 'steps' styles can be used with 'splot's. If an inappropriate style is specified, it will be changed to 'points'.

The above caveat does not apply to 'plot with labels', for which the third column specifies a data source rather than coordinate information. See 'set style labels'.

For 2-d data with more than two columns, 'gnuplot' is picky about the allowed Section 2.12.2 [errorbars], page 64 and Section 2.12.3 [errorlines], page 64 styles. The Section 2.12.1.9 [using], page 61 option on the 'plot' command can be used to set up the correct columns for the style you want. (In this discussion, "column" will be used to refer both to a column in the data file and an entry in the Section 2.12.1.9 [using], page 61 list.)

For three columns, only 'xerrorbars', 'yerrorbars' (or Section 2.12.2 [errorbars], page 64), 'xerrorlines', 'yerrorlines' (or Section 2.12.3 [errorlines], page 64), 'boxes', and 'boxerrorbars' are allowed. If another plot style is used, the style will be changed to 'yerrorbars'. The 'boxerrorbars' style will calculate the boxwidth automatically.

For four columns, only 'xerrorbars', 'yerrorbars' (or Section 2.12.2 [errorbars], page 64), 'xyerrorbars', 'xerrorlines', 'yerrorlines' (or Section 2.12.3 [errorlines], page 64), 'xyerrorlines', 'boxxyerrorbars', and 'boxerrorbars' are allowed. An illegal style will be changed to 'yerrorbars'.

Five-column data allow only the 'boxerrorbars', 'financebars', and 'candlesticks' styles. An illegal style will be changed to 'boxerrorbars' before plotting.

Six- and seven-column data only allow the 'xyerrorbars', 'xyerrorlines', and 'boxxyerrorbars' styles. Illegal styles will be changed to 'xyerrorbars' before plotting.

For more information about error bars with and without lines, please see Section 2.12.3 [errorlines], page 64 and Section 2.12.2 [errorbars], page 64.

— BOXERRORBARS —

The 'boxerrorbars' style is only relevant to 2-d data plotting. It is a combination of the 'boxes' and 'yerrorbars' styles. The boxwidth will come from the fourth column if the y errors are in the form of "ydelta" and the boxwidth was not previously set equal to -2.0 (‘set boxwidth -2.0‘) or from the fifth column if the y errors are in the form of "ylow yhigh". The special case ‘boxwidth = -2.0‘ is for four-column data with y errors in the form "ylow yhigh". In this case the boxwidth will be calculated so that each box touches the adjacent boxes. The width will also be calculated in cases where three-column data are used.

The box height is determined from the y error in the same way as it is for the ‘yerrorbars‘ style—either from y-ydelta to y+ydelta or from ylow to yhigh, depending on how many data columns are provided. See also errorbar demo.

— BOXES —

The ‘boxes‘ style is only relevant to 2-d plotting. It draws a box centered about the given x coordinate from the x axis (not the graph border) to the given y coordinate. The width of the box is obtained in one of three ways. If it is a data plot and the data file has a third column, this will be used to set the width of the box. If not, if a width has been set using the Section 2.21.7 [boxwidth], page 81 command, this will be used. If neither of these is available, the width of each box will be calculated automatically so that it touches the adjacent boxes.

The interior of the boxes is drawn according to the current fillstyle. See ‘set style fill‘ for details. Alternatively a new fillstyle may be specified in the plot command.

For fillstyle ‘empty‘ the box is filled with the background color.

For fillstyle ‘solid‘ the box is filled with a solid rectangle of the current drawing color. There is an optional parameter <density> that controls the fill density; it runs from 0 (background color) to 1 (current drawing color).

For fillstyle ‘pattern‘ the box is filled in the current drawing color with a pattern, if supported by the terminal driver.

Examples:

To plot a data file with solid filled boxes with a small vertical space separating them (bargraph):

```
set boxwidth 0.9 relative
set style fill solid 1.0
plot 'file.dat' with boxes
```

To plot a sine and a cosine curve in pattern-filled boxes style:

```
set style fill pattern
plot sin(x) with boxes, cos(x) with boxes
```

The sin plot will use pattern 0; the cos plot will use pattern 1. Any additional plots would cycle through the patterns supported by the terminal driver.

To specify explicit fillstyles for each dataset:

```
plot 'file1' with boxes fs solid 0.25, \
     'file2' with boxes fs solid 0.50, \
     'file3' with boxes fs solid 0.75, \
     'file4' with boxes fill pattern 1, \
     'file5' with boxes fill empty
```

Currently only the following terminal drivers support fillstyles other than ‘empty‘: x11, windows, pm, postscript, fig, pbm, png, gif, hpdj, hppj, hpljii, hp500c, jpeg, nec_cp6, epson_180dpi, epson_60dpi, epson_lx800, okidata, starc and tandy_60dpi. The BeOS driver (‘be‘) is untested.

— BOXXYERRORBARS —

The ‘boxxyerrorbars‘ style is only relevant to 2-d data plotting. It is a combination of the ‘boxes‘ and ‘xyerrorbars‘ styles.

The box width and height are determined from the x and y errors in the same way as they are for the ‘xyerrorbars‘ style—either from xlow to xhigh and from ylow to yhigh, or from x-xdelta to x+xdelta and from y-ydelta to y+ydelta , depending on how many data columns are provided.

If filled-box support is present, then the interior of the boxes is drawn according to the current fillstyle. See ‘set style fill‘ and ‘boxes‘ for details. Alternatively a new fillstyle may be specified in the plot command.

— CANDLESTICKS —

The ‘candlesticks‘ style can be used for 2-d data plotting of financial data or for generating box-and-whisker plots of statistical data. Five columns of data are required; in order, these should be the x coordinate (most likely a date) and the opening, low, high, and closing prices. The symbol is a rectangular box, centered horizontally at the x coordinate and limited vertically by the opening and closing prices. A vertical line segment at the x coordinate extends up from the top of the rectangle to the high price and another down to the low. The vertical line will be unchanged if the low and high prices are interchanged.

The width of the rectangle can be controlled by the Section 2.21.7 [boxwidth], page 81 command. For backwards compatibility with earlier gnuplot versions, when the boxwidth parameter has not been set then the width of the candlestick rectangle is controlled by ‘set bars <width>‘.

By default the vertical line segments have no crossbars at the top and bottom. If you want crossbars, which are typically used for box-and-whisker plots, then add the keyword ‘whiskerbars‘ to the plot command.

By default the rectangle is empty if (open > close), and filled with three vertical bars if (close > open). If filled-boxes support is present, then the rectangle is colored according to ‘set style fill <fillstyle>‘. See Section 2.21.4 [bars], page 79 and ‘financebars‘. See also finance demos .

Note: To place additional symbols, such as the median value, on a box-and-whisker plot requires additional plot commands as in this example:

```
# Data columns: X Min 1stQuartile Median 3rdQuartile Max
set bars 4.0
set style fill empty
plot 'stat.dat' using 1:3:2:6:5 with candlesticks title 'Quar-
tiles', \
     ''         using 1:4:4:4:4 with candlesticks lt -1 notitle
```

```
# Plot with crossbars on the whiskers
plot 'stat.dat' using 1:3:2:6:5 with candlesticks whiskerbars
```

See Section 2.21.7 [boxwidth], page 81, Section 2.21.4 [bars], page 79 and 'set style fill'.

— DOTS —

The 'dots' style plots a tiny dot at each point; this is useful for scatter plots with many points. For some terminals (post, pdf) the size of the dot can be controlled by changing the linewidth.

— FILLEDCURVES —

The 'filledcurves' style is only relevant to 2-d plotting. Three variants are possible. The first two variants require either a function or two columns of input data, and may be further modified by the options listed below. The first variant, 'closed', treats the curve itself as a closed polygon. This is the default.

The second variant is to fill the area between the curve and a given axis, a horizontal or vertical line, or a point.

The third variant requires three columns of input data: the x coordinate and two y coordinates corresponding to two curves sampled at the same set of x coordinates; the area between the two curves is filled.

Syntax:

```
set style [data | function] filledcurves [option]
plot ... with filledcurves [option]
```

where the option can be

```
[closed | {above | below} {x1 | x2 | y1 | y2}[=<a>] | xy=<x>,<y>]
```

The first two plot variants can be further modified by the options

```
    filledcurves closed   ... just filled closed curve,
    filledcurves x1       ... x1 axis,
    filledcurves x2       ... x2 axis, etc for y1 and y2 axes,
    filledcurves y1=0     ... line y=0 (at y1 axis) ie paral-
lel to x1 axis,
    filledcurves y2=42    ... line y=42 (at y2 axis) ie paral-
lel to x2, etc,
    filledcurves xy=10,20 ... point 10,20 of x1,y1 axes (arc-
like shape).
```

Example of filling the area between two input curves. fill between curves demo.

```
plot 'data' using 1:2:3 with filledcurves
```

The 'above' and 'below' options apply both to commands of the form

```
... filledcurves above {x1|x2|y1|y2}=<val>
```

and to commands of the form

```
... using 1:2:3 with filledcurves below
```

In either case the option limits the filled area to one side of the bounding line or curve.

Note: Not all terminal types support this plotting mode.

Zoom of a filled curve drawn from a datafile may produce empty or incorrect area because gnuplot is clipping points and lines, and not areas.

If the values of <a>, <x>, <y> are out of the drawing boundary, then they are moved to the graph boundary. Then the actually filled area in the case of option xy=<x>,<y> will depend on xrange and yrange.

— FINANCEBARS —

The 'financebars' style is only relevant for 2-d data plotting of financial data. Five columns of data are required; in order, these should be the x coordinate (most likely a date) and the opening, low, high, and closing prices. The symbol is a vertical line segment, located horizontally at the x coordinate and limited vertically by the high and low prices. A horizontal tic on the left marks the opening price and one on the right marks the closing price. The length of these tics may be changed by Section 2.21.4 [bars], page 79. The symbol will be unchanged if the high and low prices are interchanged. See Section 2.21.4 [bars], page 79 and 'candlesticks', and also the finance demo.

— FSTEPS —

The 'fsteps' style is only relevant to 2-d plotting. It connects consecutive points with two line segments: the first from (x1,y1) to (x1,y2) and the second from (x1,y2) to (x2,y2). See also steps demo.

— HISTEPS —

The 'histeps' style is only relevant to 2-d plotting. It is intended for plotting histograms. Y-values are assumed to be centered at the x-values; the point at x1 is represented as a horizontal line from ((x0+x1)/2,y1) to ((x1+x2)/2,y1). The lines representing the end points are extended so that the step is centered on at x. Adjacent points are connected by a vertical line at their average x, that is, from ((x1+x2)/2,y1) to ((x1+x2)/2,y2).

If Section 2.21.3 [autoscale], page 77 is in effect, it selects the xrange from the data rather than the steps, so the end points will appear only half as wide as the others. See also steps demo.

'histeps' is only a plotting style; 'gnuplot' does not have the ability to create bins and determine their population from some data set.

— HISTOGRAMS —

The 'histograms' style is only relevant to 2-d plotting. It produces a bar chart from a sequence of data columns in parallel. Each element of the 'plot' command must specify a single input data source (e.g. one column of the input file), possibly with associated tic values or key titles. Four styles of histogram layout are currently supported.

```
set style histogram clustered {gap <gapsize>}
set style histogram errorbars {gap <gapsize>} {<linewidth>}
set style histogram rowstacked
set style histogram columnstacked
```

The default style corresponds to ‘set style histogram clustered gap 2‘. In this style, each set of parallel data values is collected into a group of boxes clustered at the x-axis coordinate corresponding to their sequential position (row #) in the selected datafile columns. Thus if <n> datacolumns are selected, the first cluster is centered about x=1, and contains <n> boxes whose heights are taken from the first entry in the corresponding <n> data columns. This is followed by a gap and then a second cluster of boxes centered about x=2 corresponding to the second entry in the respective data columns, and so on. The default gap width of 2 indicates that the empty space between clusters is equivalent to the width of 2 boxes. All boxes derived from any one column are given the same fill color and/or pattern (see ‘set style fill‘).

Each cluster of boxes is derived from a single row of the input data file. It is common in such input files that the first element of each row is a label. Labels from this column may be placed along the x-axis underneath the appropriate cluster of boxes with the ‘xticlabels‘ option to Section 2.12.1.9 [using], page 61.

The Section 2.12.2 [errorbars], page 64 style is very similar to the ‘clustered‘ style, except that it requires two columns of input for each entry. The first column is treated as the height (y-value) of that box, exactly as for the ‘clustered‘ style. The second column is treated as an error magnitude, and used to generate a vertical error bar at the top of the box. The appearance of the error bar is controlled by the current value of Section 2.21.4 [bars], page 79 and by the optional <linewidth> specification.

Two styles of stacked histogram are supported, chosen by the command ‘set style histogram {rowstacked|columnstacked}‘. In these styles the data values from the selected columns are collected into stacks of boxes. The default stacking mode is ‘rowstacked‘.

The ‘rowstacked‘ style places a box resting on the x-axis for each data value in the first selected column; the first data value results in a box a x=1, the second at x=2, and so on. Boxes corresponding to the second and subsequent data columns are layered on top of these, resulting in a stack of boxes at x=1 representing the first data value from each column, a stack of boxes at x=2 representing the second data value from each column, and so on. All boxes derived from any one column are given the same fill color and/or pattern (see ‘set style fill‘).

The ‘columnstacked‘ style is similar, except that each stack of boxes is built up from a single data column. Each data value from the first specified column yields a box in the stack at x=1, each data value from the second specified column yields a box in the stack at x=2, and so on. In this style the color of each box is taken from the row number, rather than the column number, of the corresponding data field.

Box widths may be modified using the Section 2.21.7 [boxwidth], page 81 command. Box fill styles may be set using the ‘set style fill‘ command.

Histograms always use the x1 axis, but may use either y1 or y2. If a plot contains both histograms and other plot styles, the non-histogram plot elements may use either the x1 or the x2 axis.

Examples:

To plot a data file containing multiple columns of data as a histogram of clustered boxes (the default style):

```
set boxwidth 0.9 relative
set style data histograms
set style fill solid 1.0 border -1
plot 'file.dat' using 2, '' using 4, '' using 6
```

This will produce a plot with clusters of three boxes (vertical bars) centered at each integral value on the x axis. If the first column of the input file contains labels, they may be placed along the x-axis using the variant command

```
plot 'file.dat' using 2, '' using 4, '' using 6:xticlabels(1)
```

If the file contains both a magnitude and an error estimate for each value, then error bars can be added to the plot. The following commands will add error bars extending from (y-<error>) to (y+<error>), capped by horizontal bar ends drawn the same width as the box itself. The error bars and bar ends are drawn in black with linewidth 2.

```
set bars fullwidth
set style histogram errorbars gap 2 lt -1 lw 2
plot 'file.dat' using 2:3, '' using 4:5, '' us-
ing 6:7:xticlabels(1)
```

To plot the same data as a rowstacked histogram:

```
set style histogram rows
plot 'file.dat' using 2, '' using 4, '' using 6:xtic(1)
```

This will produce a plot in which each vertical bar contains a stack of three segments, corresponding in height to the values found in columns 2, 4 and 6 of the datafile.

Finally, the commands

```
set style histogram columnstacked
plot 'file.dat' using 2, '' using 4, '' using 6
```

will produce three vertical stacks. The stack at x=1 will contain a box for each entry in column 2 of the datafile. The stack at x=2 will contain a box for each parallel entry in column 4 of the datafile, and the stack at x=3 a box for each entry of column 6. Because this interchanges gnuplot's usual interpretation of input rows and columns, the specification of key titles and x-axis tic labels must also be modified.

```
set style histogram columnstacked
plot '' u 5:key(1)            # uses first column to gener-
ate key titles
plot '' u 5 title columnhead  # uses first row to gener-
ate xtic labels
```

— NEWHISTOGRAM —

More than one set of histograms can appear in a single plot. In this case you can force a gap between them, and a separate label for each set, by using the plot command 'newhistogram { "title" } { <linestyle> }'. For example

```
set style histogram  cluster
plot newhistogram "Set A", 'a' using 1, '' using 2, '' us-
ing 3, \
     newhistogram "Set B", 'b' using 1, '' using 2, '' using 3
```

The labels "Set A" and "Set B" will appear beneath the respective sets of histograms, under the overall x axis label.

The newhistogram command can also be used to force histogram coloring to begin with a specific color (linetype). By default colors will continue to increment successively even across histogram boundaries. Here is an example using the same coloring for multiple histograms

```
plot newhistogram "Set A" lt 4, 'a' using 1, '' using 2, '' us-
ing 3, \
     newhistogram "Set B" lt 4, 'b' using 1, '' using 2, '' us-
ing 3
```

— IMAGE —

The ‘image‘ style is intendend for plotting 2D images. It may be used for both ‘plot‘ and ‘splot‘ provided the 3D data (x,y,value) or projected 4D data (x,y,z,value) forms a valid grid. A valid grid is one in which the data in the viewing plane is equidistant along two, not necessarily orthogonal, directions. In other words, groups of four adjacent points must form the same size parallelogram. The variable ‘value‘ in the tuples represent a palette color (gray value) for indexing in the current palette.

If the data points form a valid rectangular grid, i.e., a valid grid for which the directions are orthogonal and lie along the x and y axes of the viewing plane, the ‘image‘ style will attempt to create a properly positioned and scaled data matrix to match the plot borders for those terminals supporting palettes and images. The result is efficient storage and fast refresh. However, for terminal drivers not supporting palettes and images, or those not yet implemented, the ‘image‘ style will revert to drawing filled rectangular boxes for pixels, which is not as efficient. Furthermore, general parallelogram-shaped images have filled parallelograms for pixels.

The coordinate of each data point of an image will lie at the center of a pixel. That is, an M x N set of data will form an image with M x N pixels. This is slightly different than pm3d elements where an M x N set of data will form a surface of (M-1) x (N-1) elements. The scan directions for the image data grid can be any of eight possible combinations.

Currently only the following terminal drivers support data matrix images:

x11 - Pixels are either repeated or decimated to fit the x11 resolution;

```
no other processing (filtering) is done.  Thus, alias-
ing may occur
when decimating images having high spatial frequency content.
```

postscript (pslatex, epslatex, pstex) - Image is copied in its original

```
resolution, and sample interpolation is turned off.
```

png - Output is dependent on the installed version of libgd.

```
gd 1.8.4  No truecolor support, but the driver function works.
gd 2.0.4  Truecolor works, but if truecolor is not selected
          the image comes out blank.
gd 2.0.9  Truecolor works, non-truecolor also works.
```

aqua pdf, svg

See also ‘rgbimage‘.

— IMPULSES —

The ‘impulses‘ style displays a vertical line from the x axis (not the graph border), or from the grid base for ‘splot‘, to each point.

— LABELS —

The ‘labels‘ style is available only if gnuplot is built with configuration option –enable-datastrings. For a 2-D plot with labels you must specify 3 input data columns; the text string found in the third column is printed at the X and Y coordinates generated by the first two column specifiers. The font, color, rotation angle and other properties of the printed text may be specified as additional command options (see Section 2.21.29 [label], page 104). The example below will generate a 2-D plot with text labels taken from column 4 of the input file (‘tc lt 2‘ is shorthand for ‘textcolor linetype 2‘, which is green).

```
  plot 'datafile' using 1:(0.5 * $2):4 with labels font "ar-
ial,11" tc lt 2
```

The ‘labels‘ style can also be used in 3-D plots. In this case four input column specifiers are required, corresponding to X Y Z and text.

```
splot 'datafile' using 1:2:3:4 with labels
```

See also ‘datastrings‘, ‘set style data‘.

— LINES —

The ‘lines‘ style connects adjacent points with straight line segments. See also ‘linetype‘, ‘linewidth‘, and ‘linestyle‘.

— LINESPOINTS —

The ‘linespoints‘ style does both ‘lines‘ and ‘points‘, that is, it draws a small symbol at each point and then connects adjacent points with straight line segments. The command Section 2.21.51 [pointsize], page 126 may be used to change the size of the points. See Section 2.21.51 [pointsize], page 126 for its usage.

‘linespoints‘ may be abbreviated ‘lp‘.

— POINTS —

The ‘points‘ style displays a small symbol at each point. The command Section 2.21.51 [pointsize], page 126 may be used to change the size of the points. See Section 2.21.51 [pointsize], page 126 for its usage.

— STEPS —

The ‘steps‘ style is only relevant to 2-d plotting. It connects consecutive points with two line segments: the first from (x1,y1) to (x2,y1) and the second from (x2,y1) to (x2,y2). See also steps demo.

— RGBIMAGE —

The ‘rgbimage‘ style is intended for plotting 2D images and is similar in concept to ‘image‘. See ‘image‘ for details. The difference is that 5D data (x,y,r,g,b) for ‘plot‘ and 6D data (x,y,z,r,g,b) for ‘splot‘ describe the coordinates and color components of an image. Currently, color images are only possible when the sampling grid is rectangular and oriented with the view plane. General orientations must be done with filled polygons which only support palette table lookup, not primary colors.

See also ‘image‘.

— VECTORS —

The 2D ‘vectors‘ style draws a vector from (x,y) to (x+xdelta,y+ydelta). Thus it requires four columns of data. It also draws a small arrowhead at the end of the vector. The 3D ‘vectors‘ style is similar, but requires six columns of data. splot with vectors is supported only for ‘set mapping cartesian‘. The keywords "with vectors" may be followed by arrow style specifications. See ‘arrowstyle‘ for more details.

Example:

```
      plot 'file.dat' using 1:2:3:4 with vectors head filled lt 2
      splot 'file.dat' using 1:2:3:(1):(1):(1) with vec-
tors filled head lw 2
```

‘set clip one‘ and ‘set clip two‘ affect vectors drawn in 2D. Please see Section 2.21.9 [clip], page 82 and ‘arrowstyle‘.

— XERRORBARS —

The ‘xerrorbars‘ style is only relevant to 2-d data plots. ‘xerrorbars‘ is like ‘dots‘, except that a horizontal error bar is also drawn. At each point (x,y), a line is drawn from (xlow,y) to (xhigh,y) or from (x-xdelta,y) to (x+xdelta,y), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless Section 2.21.4 [bars], page 79 is used—see Section 2.21.4 [bars], page 79 for details).

— XYERRORBARS —

The ‘xyerrorbars‘ style is only relevant to 2-d data plots. ‘xyerrorbars‘ is like ‘dots‘, except that horizontal and vertical error bars are also drawn. At each point (x,y), lines are drawn from (x,y-ydelta) to (x,y+ydelta) and from (x-xdelta,y) to (x+xdelta,y) or from (x,ylow) to (x,yhigh) and from (xlow,y) to (xhigh,y), depending upon the number of data columns provided. A tic mark is placed at the ends of the error bar (unless Section 2.21.4 [bars], page 79 is used—see Section 2.21.4 [bars], page 79 for details).

If data are provided in an unsupported mixed form, the Section 2.12.1.9 [using], page 61 filter on the ‘plot‘ command should be used to set up the appropriate form. For example, if the data are of the form (x,y,xdelta,ylow,yhigh), then you can use

```
plot 'data' using 1:2:($1-$3):($1+$3):4:5 with xyerrorbars
```

— YERRORBARS —

The 'yerrorbars' (or Section 2.12.2 [errorbars], page 64) style is only relevant to 2-d data plots. 'yerrorbars' is like 'points', except that a vertical error bar is also drawn. At each point (x,y), a line is drawn from (x,y-ydelta) to (x,y+ydelta) or from (x,ylow) to (x,yhigh), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless Section 2.21.4 [bars], page 79 is used—see Section 2.21.4 [bars], page 79 for details). See also errorbar demo.

— XERRORLINES —

The 'xerrorlines' style is only relevant to 2-d data plots. 'xerrorlines' is like 'linespoints', except that a horizontal error line is also drawn. At each point (x,y), a line is drawn from (xlow,y) to (xhigh,y) or from (x-xdelta,y) to (x+xdelta,y), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless Section 2.21.4 [bars], page 79 is used—see Section 2.21.4 [bars], page 79 for details).

— XYERRORLINES —

The 'xyerrorlines' style is only relevant to 2-d data plots. 'xyerrorlines' is like 'linespoints', except that horizontal and vertical error bars are also drawn. At each point (x,y), lines are drawn from (x,y-ydelta) to (x,y+ydelta) and from (x-xdelta,y) to (x+xdelta,y) or from (x,ylow) to (x,yhigh) and from (xlow,y) to (xhigh,y), depending upon the number of data columns provided. A tic mark is placed at the ends of the error bar (unless Section 2.21.4 [bars], page 79 is used—see Section 2.21.4 [bars], page 79 for details).

If data are provided in an unsupported mixed form, the Section 2.12.1.9 [using], page 61 filter on the 'plot' command should be used to set up the appropriate form. For example, if the data are of the form (x,y,xdelta,ylow,yhigh), then you can use

```
plot 'data' using 1:2:($1-$3):($1+$3):4:5 with xyerrorlines
```

— YERRORLINES —

The 'yerrorlines' (or Section 2.12.3 [errorlines], page 64) style is only relevant to 2-d data plots. 'yerrorlines' is like 'linespoints', except that a vertical error line is also drawn. At each point (x,y), a line is drawn from (x,y-ydelta) to (x,y+ydelta) or from (x,ylow) to (x,yhigh), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless Section 2.21.4 [bars], page 79 is used—see Section 2.21.4 [bars], page 79 for details). See also errorbar demo.

2.21.59 surface

The command Section 2.21.59 [surface], page 144 controls the display of surfaces by 'splot'.

Syntax:

```
set surface
unset surface
show surface
```

The surface is drawn with the style specified by Section 2.12.7 [with], page 68, or else the appropriate style, data or function.

Whenever Section 2.21.59 [surface], page 144 is issued, 'splot' will not draw points or lines corresponding to the function or data file points. Contours may still be drawn on the surface, depending on the Section 2.21.12 [contour], page 86 option. 'unset surface; set contour base‘ is useful for displaying contours on the grid base. See also Section 2.21.12 [contour], page 86.

2.21.60 table

When Section 2.21.60 [table], page 145 mode is enabled, ‘plot‘ and ‘splot‘ commands print out a multicolumn ASCII table of X Y {Z} R values rather than creating an actual plot on the current terminal. The character R takes on one of three values: "i" if the point is in the active range, "o" if it is out-of-range, or "u" if it is undefined. The data format is determined by the format of the axis labels (see ‘set format‘), and the columns are separated by single spaces. This can be useful if you want to generate contours and then save them for further use, perhaps for plotting with ‘plot‘; see Section 2.21.12 [contour], page 86 for example. The same method can be used to save interpolated data (see Section 2.21.56 [samples], page 128 and Section 2.21.16 [dgrid3d], page 90).

Syntax:

```
set table {"outfile"}
plot <whatever>
unset table
```

Tabular output is written to the named file, if any, otherwise it is written to the current value of Section 2.21.46 [output], page 115. You must explicitly Section 2.21.60 [table], page 145 in order to go back to normal plotting on the current terminal.

2.21.61 terminal

‘gnuplot‘ supports many different graphics devices. Use ‘set terminal‘ to tell ‘gnuplot‘ what kind of output to generate. Use Section 2.21.46 [output], page 115 to redirect that output to a file or device.

Syntax:

```
set terminal {<terminal-type> | push | pop}
show terminal
```

If <terminal-type> is omitted, ‘gnuplot‘ will list the available terminal types. <terminal-type> may be abbreviated.

If both ‘set terminal‘ and Section 2.21.46 [output], page 115 are used together, it is safest to give ‘set terminal‘ first, because some terminals set a flag which is needed in some operating systems.

Several terminals have many additional options. For example, see ‘png‘, or Section 2.21.50.6 [postscript], page 126. The options used by a previous invocation ‘set term <term> <options>‘ of a given ‘<term>‘ are remembered, thus subsequent ‘set term <term>‘ does not reset them. This helps in printing, for instance, when switching among different terminals—previous options don’t have to be repeated.

The command ‘set term push‘ remembers the current terminal including its settings while ‘set term pop‘ restores it. This is equivalent to ‘save term‘ and ‘load term‘, but without accessing the filesystem. Therefore they can be used to achieve platform independent restoring of the terminal after printing, for instance. After gnuplot’s startup, the default terminal or that from ‘startup‘ file is pushed automatically. Therefore portable scripts can rely that ‘set term pop‘ restores the default terminal on a given platform unless another terminal has been pushed explicitly.

For a complete list of available terminal types, see ‘terminal‘.

2.21.62 termoption

The Section 2.21.62 [termoption], page 146 command allows you to change the behaviour of the current terminal without requiring a new ‘set terminal‘ command. Only one option can be changed per command, and only a small number of options can be changed this way. Currently the only options accepted are

```
set termoption {no}enhanced
set termoption font "<fontname>{,<fontsize>}"
```

2.21.63 tics

Control of the major (labelled) tics on all axes at once is possible with the ‘set tics‘ command.

Fine control of the major (labelled) tics on all axes at once is possible with the ‘set tics‘ command. The tics may be turned off with the ‘unset tics‘ command, and may be turned on (the default state) with ‘set tics‘. Similar commands (by preceding ’tics’ by the axis name) control the major tics on a single axis.

Syntax:

```
set tics {axis | border} {{no}mirror}
         {in | out} {scale {default | <major> {,<minor>}}}
         {{no}rotate {by <ang>}} {offset <offset> | nooffset}
         { font "name{,<size>}" }
         { textcolor <colorspec> }
unset tics
show tics
```

All specified options apply to all axes, i.e., x, y, z, x2, y2, and cb.

‘axis‘ or Section 2.21.6 [border], page 79 tells ‘gnuplot‘ to put the tics (both the tics themselves and the accompanying labels) along the axis or the border, respectively. If the axis is very close to the border, the ‘axis‘ option will move the tic labels to outside the border in case the border is printed (see Section 2.21.6 [border], page 79). The relevant margin settings will usually be sized badly by the automatic layout algorithm in this case.

‘mirror‘ tells ‘gnuplot‘ to put unlabelled tics at the same positions on the opposite border. ‘nomirror‘ does what you think it does.

‘in‘ and ‘out‘ change the tic marks to be drawn inwards or outwards.

With ‘scale‘, the size of the tic marks can be adjusted. If <minor> is not specified, it is 0.5*<major>. The default size 1.0 for major tics and 0.5 for minor tics is requested by ‘scale default‘.

‘rotate‘ asks ‘gnuplot‘ to rotate the text through 90 degrees, which will be done if the terminal driver in use supports text rotation. ‘norotate‘ cancels this. ‘rotate by <ang>‘ asks for rotation by <ang> degrees, supported by some terminal types.

The defaults are ‘border mirror norotate‘ for tics on the x and y axes, and ‘border nomirror norotate‘ for tics on the x2 and y2 axes. For the z axis, the default is ‘nomirror‘.

The <offset> is specified by either x,y or x,y,z, and may be preceded by ‘first‘, ‘second‘, ‘graph‘, ‘screen‘, or ‘character‘ to select the coordinate system. <offset> is the offset of the tics texts from their default positions, while the default coordinate system is ‘character‘. See ‘coordinates‘ for details. ‘nooffset‘ switches off the offset.

‘set tics‘ with no options restores to place tics inwards. Every other options are retained.

See also Section 2.21.88 [xtics], page 155 for more control of major (labelled) tic marks and Section 2.21.40 [mxtics], page 113 for control of minor tic marks. These commands provide control at a axis by axis basis.

2.21.64 ticslevel

See Section 2.21.89 [xyplane], page 159.

2.21.65 ticscale

The Section 2.21.65 [ticscale], page 147 command is deprecated, use ‘set tics scale‘ instead.

2.21.66 timestamp

The command Section 2.21.66 [timestamp], page 147 places the time and date of the plot in the left margin.

Syntax:

```
set timestamp {"<format>"} {top|bottom} {{no}rotate}
              {offset {<xoff>}{,<yoff>}} {font "<fontspec>"}
unset timestamp
show timestamp
```

The format string allows you to choose the format used to write the date and time. Its default value is what asctime() uses: "%a %b %d %H:%M:%S %Y" (weekday, month name, day of the month, hours, minutes, seconds, four-digit year). With ‘top‘ or ‘bottom‘ you can place the timestamp at the top or bottom of the left margin (default: bottom). ‘rotate‘ lets you write the timestamp vertically, if your terminal supports vertical text. The constants <xoff> and <yoff> are offsets that let you adjust the position more finely. <font> is used to specify the font with which the time is to be written.

The abbreviation ‘time‘ may be used in place of Section 2.21.66 [timestamp], page 147.

Example:

```
set timestamp "%d/%m/%y %H:%M" offset 80,-2 font "Helvetica"
```

See Section 2.21.67 [timefmt], page 147 for more information about time format strings.

2.21.67 timefmt

This command applies to timeseries where data are composed of dates/times. It has no meaning unless the command ‘set xdata time‘ is given also.

Syntax:

```
set timefmt "<format string>"
show timefmt
```

The string argument tells ‘gnuplot‘ how to read timedata from the datafile. The valid formats are:

```
Format       Explanation
%d           day of the month, 1--31
%m           month of the year, 1--12
%y           year, 0--99
%Y           year, 4-digit
%j           day of the year, 1--365
%H           hour, 0--24
%M           minute, 0--60
%s           seconds since the Unix epoch (1970-01-
01, 00:00 UTC)
%S           second, 0--60
%b           three-character abbrevia-
tion of the name of the month
%B           name of the month
```

Any character is allowed in the string, but must match exactly. \t (tab) is recognized. Backslash-octals (\nnn) are converted to char. If there is no separating character between the time/date elements, then %d, %m, %y, %H, %M and %S read two digits each, %Y reads four digits and %j reads three digits. %b requires three characters, and %B requires as many as it needs.

Spaces are treated slightly differently. A space in the string stands for zero or more whitespace characters in the file. That is, "%H %M" can be used to read "1220" and "12 20" as well as "12 20".

Each set of non-blank characters in the timedata counts as one column in the ‘using n:n‘ specification. Thus ‘11:11 25/12/76 21.0‘ consists of three columns. To avoid confusion, ‘gnuplot‘ requires that you provide a complete Section 2.12.1.9 [using], page 61 specification if your file contains timedata.

Since ‘gnuplot‘ cannot read non-numerical text, if the date format includes the day or month in words, the format string must exclude this text. But it can still be printed with the "%a", "%A", "%b", or "%B" specifier: see ‘set format‘ for more details about these and other options for printing timedata. (‘gnuplot‘ will determine the proper month and weekday from the numerical values.)

See also Section 2.21.83 [xdata], page 151 and ‘Time/date‘ for more information.

Example:

```
set timefmt "%d/%m/%Y\t%H:%M"
```

tells ‘gnuplot‘ to read date and time separated by tab. (But look closely at your data—what began as a tab may have been converted to spaces somewhere along the line; the format string must match what is actually in the file.) See also time data demo.

2.21.68 title

The ‘set title‘ command produces a plot title that is centered at the top of the plot. ‘set title‘ is a special case of Section 2.21.29 [label], page 104.

Syntax:

```
      set title {"<title-text>"} {offset <off-
set>} {font "<font>{,<size>}"}
                {{textcolor | tc} {lt <line_type> | de-
fault}} {{no}enhanced}
      show title
```

If <offset> is specified the title is moved by the given offset where <offset> is specified by either x,y or x,y,z, and may be preceded by ‘first‘, ‘second‘, ‘graph‘, ‘screen‘, or ‘character‘ to select the coordinate system. See ‘coordinates‘ for details. By default, the ‘character‘ coordinate system is used. For example, "‘set title 0,-1‘" will change only the y offset of the title, moving the title down by roughly the height of one character.

<font> is used to specify the font with which the title is to be written; the units of the font <size> depend upon which terminal is used.

‘textcolor lt <n>‘ sets the text color to that of line type <n>.

‘noenhanced‘ requests that the title not be processed by the enhanced text mode parser, even if enhanced text mode is currently active.

‘set title‘ with no parameters clears the title.

See ‘syntax‘ for details about the processing of backslash sequences and the distinction between single- and double-quotes.

2.21.69 tmargin

The command Section 2.21.69 [tmargin], page 149 sets the size of the top margin. Please see Section 2.21.36 [margin], page 109 for details.

2.21.70 trange

The Section 2.21.70 [trange], page 149 command sets the parametric range used to compute x and y values when in parametric or polar modes. Please see Section 2.21.87 [xrange], page 154 for details.

2.21.71 urange

The Section 2.21.71 [urange], page 149 and Section 2.21.75 [vrange], page 150 commands set the parametric ranges used to compute x, y, and z values when in ‘splot‘ parametric mode. Please see Section 2.21.87 [xrange], page 154 for details.

2.21.72 variables

The Section 2.21.72 [variables], page 149 command lists all user-defined variables and their values.

Syntax:

```
      show variables
```

2.21.73 version

The Section 2.21.73 [version], page 150 command lists the version of gnuplot being run, its last modification date, the copyright holders, and email addresses for the FAQ, the gnuplot-info mailing list, and reporting bugs–in short, the information listed on the screen when the program is invoked interactively.

Syntax:

```
show version {long}
```

When the ‘long‘ option is given, it also lists the operating system, the compilation options used when ‘gnuplot‘ was installed, the location of the help file, and (again) the useful email addresses.

2.21.74 view

The Section 2.21.74 [view], page 150 command sets the viewing angle for ‘splot‘s. It controls how the 3-d coordinates of the plot are mapped into the 2-d screen space. It provides controls for both rotation and scaling of the plotted data, but supports orthographic projections only. It supports both 3D projection or orthogonal 2D projection into a 2D plot-like map.

Syntax:

```
set view { <rot_x>{,{<rot_z>}{,{<scale>}{,<scale_z>}}} | map }
show view
```

where <rot_x> and <rot_z> control the rotation angles (in degrees) in a virtual 3-d coordinate system aligned with the screen such that initially (that is, before the rotations are performed) the screen horizontal axis is x, screen vertical axis is y, and the axis perpendicular to the screen is z. The first rotation applied is <rot_x> around the x axis. The second rotation applied is <rot_z> around the new z axis.

Command ‘set view map‘ is used to represent the drawing as a map. It can be used for Section 2.21.12 [contour], page 86 plots, or for color pm3d maps. In the latter, take care that you properly use Section 2.21.114 [zrange], page 162 and Section 2.21.118 [cbrange], page 162 for input data point filtering and color range scaling, respectively.

<rot_x> is bounded to the [0:180] range with a default of 60 degrees, while <rot_z> is bounded to the [0:360] range with a default of 30 degrees. <scale> controls the scaling of the entire ‘splot‘, while <scale_z> scales the z axis only. Both scales default to 1.0.

Examples:

```
set view 60, 30, 1, 1
set view ,,0.5
```

The first sets all the four default values. The second changes only scale, to 0.5.

See also Section 2.21.64 [ticslevel], page 147.

2.21.75 vrange

The Section 2.21.71 [urange], page 149 and Section 2.21.75 [vrange], page 150 commands set the parametric ranges used to compute x, y, and z values when in ‘splot‘ parametric mode. Please see Section 2.21.87 [xrange], page 154 for details.

2.21.76 x2data

The Section 2.21.76 [x2data], page 151 command sets data on the x2 (top) axis to timeseries (dates/times). Please see Section 2.21.83 [xdata], page 151.

2.21.77 x2dtics

The Section 2.21.77 [x2dtics], page 151 command changes tics on the x2 (top) axis to days of the week. Please see Section 2.21.84 [xdtics], page 152 for details.

2.21.78 x2label

The Section 2.21.78 [x2label], page 151 command sets the label for the x2 (top) axis. Please see Section 2.21.85 [xlabel], page 152.

2.21.79 x2mtics

The Section 2.21.79 [x2mtics], page 151 command changes tics on the x2 (top) axis to months of the year. Please see Section 2.21.86 [xmtics], page 153 for details.

2.21.80 x2range

The Section 2.21.80 [x2range], page 151 command sets the horizontal range that will be displayed on the x2 (top) axis. Please see Section 2.21.87 [xrange], page 154 for details.

2.21.81 x2tics

The Section 2.21.81 [x2tics], page 151 command controls major (labelled) tics on the x2 (top) axis. Please see Section 2.21.88 [xtics], page 155 for details.

2.21.82 x2zeroaxis

The Section 2.21.82 [x2zeroaxis], page 151 command draws a line at the origin of the x2 (top) axis (y2 = 0). For details, please see Section 2.21.111 [zeroaxis], page 161.

2.21.83 xdata

This command sets the datatype on the x axis to time/date. A similar command does the same thing for each of the other axes.

Syntax:

```
set xdata {time}
show xdata
```

The same syntax applies to Section 2.21.98 [ydata], page 160, Section 2.21.105 [zdata], page 160, Section 2.21.76 [x2data], page 151, Section 2.21.91 [y2data], page 159 and Section 2.21.108 [cbdata], page 161.

The 'time' option signals that the datatype is indeed time/date. If the option is not specified, the datatype reverts to normal.

See Section 2.21.67 [timefmt], page 147 to tell gnuplot how to read date or time data. The time/date is converted to seconds from start of the century. There is currently only one timefmt, which implies that all the time/date columns must conform to this format. Specification of ranges should be supplied as quoted strings according to this format to avoid interpretation of the time/date as an expression.

The function 'strftime' (type "man strftime" on unix to look it up) is used to print tic-mark labels. ‘gnuplot‘ tries to figure out a reasonable format for this unless the ‘set format x "string"‘ has supplied something that does not look like a decimal format (more than one '%' or neither %f nor %g).

See also ‘Time/date‘ for more information.

2.21.84 xdtics

The Section 2.21.84 [xdtics], page 152 commands converts the x-axis tic marks to days of the week where 0=Sun and 6=Sat. Overflows are converted modulo 7 to dates. ‘set noxdtics‘ returns the labels to their default values. Similar commands do the same things for the other axes.

Syntax:

```
set xdtics
unset xdtics
show xdtics
```

The same syntax applies to Section 2.21.99 [ydtics], page 160, Section 2.21.106 [zdtics], page 160, Section 2.21.77 [x2dtics], page 151, Section 2.21.92 [y2dtics], page 159 and Section 2.21.109 [cbdtics], page 161.

See also the ‘set format‘ command.

2.21.85 xlabel

The Section 2.21.85 [xlabel], page 152 command sets the x axis label. Similar commands set labels on the other axes.

Syntax:

```
      set xlabel {"<label>"} {offset <off-
set>} {font "<font>{,<size>}"}
                 {{textcolor | tc} {lt <line_type> | de-
fault}} {{no}enhanced}
                 {rotate by <degrees>}
      show xlabel
```

The same syntax applies to Section 2.21.78 [x2label], page 151, Section 2.21.100 [ylabel], page 160, Section 2.21.93 [y2label], page 159, Section 2.21.112 [zlabel], page 162 and Section 2.21.116 [cblabel], page 162.

If <offset> is specified the label is moved by the given offset where <offset> is specified by either x,y or x,y,z, and may be preceded by ‘first‘, ‘second‘, ‘graph‘, ‘screen‘, or ‘character‘ to select the coordinate system. See ‘coordinates‘ for details. By default, the ‘character‘ coordinate system is used. For example, "‘set xlabel offset -1,0‘" will change only the x offset of the title, moving the label roughly one character width to the left. The size of a character depends on both the font and the terminal.

<font> is used to specify the font in which the label is written; the units of the font <size> depend upon which terminal is used.

‘textcolor lt <n>‘ sets the text color to that of line type <n>.

'noenhanced' requests that the label text not be processed by the enhanced text mode parser, even if enhanced text mode is currently active.

To clear a label, put no options on the command line, e.g., "Section 2.21.93 [y2label], page 159".

The default positions of the axis labels are as follows:

xlabel: The x-axis label is centered below the bottom axis.

ylabel: The position of the y-axis label depends on the terminal, and can be one of the following three positions:

1. Horizontal text flushed left at the top left of the plot. Terminals that cannot rotate text will probably use this method. If Section 2.21.81 [x2tics], page 151 is also in use, the ylabel may overwrite the left-most x2tic label. This may be remedied by adjusting the ylabel position or the left margin.

2. Vertical text centered vertically at the left of the plot. Terminals that can rotate text will probably use this method.

3. Horizontal text centered vertically at the left of the plot. The EEPIC, LaTeX and TPIC drivers use this method. The EEPIC driver will produce a stack of characters so as not to overwrite the plot. With other drivers (such as LaTeX and TPIC), the user probably has to insert line breaks using \\ to prevent the ylabel from overwriting the plot.

zlabel: The z-axis label is centered along the z axis and placed in the space above the grid level.

cblabel: The color box axis label is centered along the box and placed below or right according to horizontal or vertical color box gradient.

y2label: The y2-axis label is placed to the right of the y2 axis. The position is terminal-dependent in the same manner as is the y-axis label.

x2label: The x2-axis label is placed above the top axis but below the plot title. It is also possible to create an x2-axis label by using new-line characters to make a multi-line plot title, e.g.,

```
set title "This is the title\n\nThis is the x2label"
```

Note that double quotes must be used. The same font will be used for both lines, of course.

The y and y2 axis labels can be explicitly rotated from their default orientation, but this applies only to 2D plots and only on terminals that support text rotation.

If you are not satisfied with the default position of an axis label, use Section 2.21.29 [label], page 104 instead–that command gives you much more control over where text is placed.

Please see 'syntax' for further information about backslash processing and the difference between single- and double-quoted strings.

2.21.86 xmtics

The Section 2.21.86 [xmtics], page 153 command converts the x-axis tic marks to months of the year where 1=Jan and 12=Dec. Overflows are converted modulo 12 to months. The tics are returned to their default labels by Section 2.21.86 [xmtics], page 153. Similar commands perform the same duties for the other axes.

Syntax:

```
set xmtics
unset xmtics
show xmtics
```

The same syntax applies to Section 2.21.79 [x2mtics], page 151, Section 2.21.101 [ymtics], page 160, Section 2.21.94 [y2mtics], page 159, Section 2.21.113 [zmtics], page 162 and Section 2.21.117 [cbmtics], page 162.

See also the ‘set format‘ command.

2.21.87 xrange

The Section 2.21.87 [xrange], page 154 command sets the horizontal range that will be displayed. A similar command exists for each of the other axes, as well as for the polar radius r and the parametric variables t, u, and v.

Syntax:

```
set xrange { [{{<min>}:{<max>}}] {{no}reverse} {{no}writeback} }
           | restore
show xrange
```

where <min> and <max> terms are constants, expressions or an asterisk to set autoscaling. If the data are time/date, you must give the range as a quoted string according to the Section 2.21.67 [timefmt], page 147 format. Any value omitted will not be changed.

The same syntax applies to Section 2.21.102 [yrange], page 160, Section 2.21.114 [zrange], page 162, Section 2.21.80 [x2range], page 151, Section 2.21.95 [y2range], page 159, Section 2.21.118 [cbrange], page 162, Section 2.21.55 [rrange], page 128, Section 2.21.70 [trange], page 149, Section 2.21.71 [urange], page 149 and Section 2.21.75 [vrange], page 150.

The ‘reverse‘ option reverses the direction of the axis, e.g., ‘set xrange [0:1] reverse‘ will produce an axis with 1 on the left and 0 on the right. This is identical to the axis produced by ‘set xrange [1:0]‘, of course. ‘reverse‘ is intended primarily for use with Section 2.21.3 [autoscale], page 77.

The ‘writeback‘ option essentially saves the range found by Section 2.21.3 [autoscale], page 77 in the buffers that would be filled by Section 2.21.87 [xrange], page 154. This is useful if you wish to plot several functions together but have the range determined by only some of them. The ‘writeback‘ operation is performed during the ‘plot‘ execution, so it must be specified before that command. To restore, the last saved horizontal range use ‘set xrange restore‘. For example,

```
set xrange [-10:10]
set yrange [] writeback
plot sin(x)
set yrange restore
replot x/2
```

results in a yrange of [-1:1] as found only from the range of sin(x); the [-5:5] range of x/2 is ignored. Executing Section 2.21.102 [yrange], page 160 after each command in the above example should help you understand what is going on.

In 2-d, Section 2.21.87 [xrange], page 154 and Section 2.21.102 [yrange], page 160 determine the extent of the axes, Section 2.21.70 [trange], page 149 determines the range of the parametric variable in parametric mode or the range of the angle in polar mode. Similarly in parametric 3-d, Section 2.21.87 [xrange], page 154, Section 2.21.102 [yrange], page 160, and Section 2.21.114 [zrange], page 162 govern the axes and Section 2.21.71 [urange], page 149 and Section 2.21.75 [vrange], page 150 govern the parametric variables.

In polar mode, Section 2.21.55 [rrange], page 128 determines the radial range plotted. <rmin> acts as an additive constant to the radius, whereas <rmax> acts as a clip to the radius—no point with radius greater than <rmax> will be plotted. Section 2.21.87 [xrange], page 154 and Section 2.21.102 [yrange], page 160 are affected—the ranges can be set as if the graph was of r(t)-rmin, with rmin added to all the labels.

Any range may be partially or totally autoscaled, although it may not make sense to autoscale a parametric variable unless it is plotted with data.

Ranges may also be specified on the 'plot' command line. A range given on the plot line will be used for that single 'plot' command; a range given by a 'set' command will be used for all subsequent plots that do not specify their own ranges. The same holds true for 'splot'.

Examples:

To set the xrange to the default:

```
set xrange [-10:10]
```

To set the yrange to increase downwards:

```
set yrange [10:-10]
```

To change zmax to 10 without affecting zmin (which may still be autoscaled):

```
set zrange [:10]
```

To autoscale xmin while leaving xmax unchanged:

```
set xrange [*:]
```

2.21.88 xtics

Fine control of the major (labelled) tics on the x axis is possible with the Section 2.21.88 [xtics], page 155 command. The tics may be turned off with the Section 2.21.88 [xtics], page 155 command, and may be turned on (the default state) with Section 2.21.88 [xtics], page 155. Similar commands control the major tics on the y, z, x2 and y2 axes.

Syntax:

```
set xtics {axis | border} {{no}mirror}
          {in | out} {scale {default | <major> {,<minor>}}}
          {{no}rotate {by <ang>}} {offset <offset> | nooffset}
```

```
              {add}
              {  autofreq
               | <incr>
               | <start>, <incr> {,<end>}
               | ({"<label>"} <pos> {<level>} {,{"<label>"}...) }
              { font "name{,<size>}" }
              { textcolor <colorspec> }
unset xtics
show xtics
```

The same syntax applies to Section 2.21.103 [ytics], page 160, Section 2.21.115 [ztics], page 162, Section 2.21.81 [x2tics], page 151, Section 2.21.96 [y2tics], page 160 and Section 2.21.119 [cbtics], page 162.

'axis' or Section 2.21.6 [border], page 79 tells 'gnuplot' to put the tics (both the tics themselves and the accompanying labels) along the axis or the border, respectively. If the axis is very close to the border, the 'axis' option will move the tic labels to outside the border. The relevant margin settings will usually be sized badly by the automatic layout algorithm in this case.

'mirror' tells 'gnuplot' to put unlabelled tics at the same positions on the opposite border. 'nomirror' does what you think it does.

'in' and 'out' change the tic marks to be drawn inwards or outwards.

With 'scale', the size of the tic marks can be adjusted. If <minor> is not specified, it is 0.5*<major>. The default size 1.0 for major tics and 0.5 for minor tics is requested by 'scale default'.

'rotate' asks 'gnuplot' to rotate the text through 90 degrees, which will be done if the terminal driver in use supports text rotation. 'norotate' cancels this. 'rotate by <ang>' asks for rotation by <ang> degrees, supported by some terminal types.

The defaults are 'border mirror norotate' for tics on the x and y axes, and 'border nomirror norotate' for tics on the x2 and y2 axes. For the z axis, the '{axis | border}' option is not available and the default is 'nomirror'. If you do want to mirror the z-axis tics, you might want to create a bit more room for them with Section 2.21.6 [border], page 79.

The <offset> is specified by either x,y or x,y,z, and may be preceded by 'first', 'second', 'graph', 'screen', or 'character' to select the coordinate system. <offset> is the offset of the tics texts from their default positions, while the default coordinate system is 'character'. See 'coordinates' for details. 'nooffset' switches off the offset.

Example:

Move xtics more closely to the plot.

```
set xtics offset 0,graph 0.05
```

Section 2.21.88 [xtics], page 155 with no options restores the default border or axis if xtics are being displayed; otherwise it has no effect. Any previously specified tic frequency or position {and labels} are retained.

Positions of the tics are calculated automatically by default or if the ‘autofreq‘ option is given; otherwise they may be specified in either of two forms:

The implicit <start>, <incr>, <end> form specifies that a series of tics will be plotted on the axis between the values <start> and <end> with an increment of <incr>. If <end> is not given, it is assumed to be infinity. The increment may be negative. If neither <start> nor <end> is given, <start> is assumed to be negative infinity, <end> is assumed to be positive infinity, and the tics will be drawn at integral multiples of <incr>. If the axis is logarithmic, the increment will be used as a multiplicative factor.

If you specify to a negative <start> or <incr> after a numerical value (e.g., ‘rotate by <angle>‘ or ‘offset <offset>‘), the parser fails because it subtracts <start> or <incr> from that value. As a workaround, specify ‘0-<start>‘ resp. ‘0-<incr>‘ in that case.

Example:

```
set xtics border offset 0,0.5 -5,1,5
```

Fails with ’invalid expression’ at the last comma.

```
set xtics border offset 0,0.5 0-5,1,5
```

or

```
set xtics offset 0,0.5 border -5,1,5
```

Sets tics at the border, tics text with an offset of 0,0.5 characters, and sets the start, increment, and end to -5, 1, and 5, as requested.

The ‘set grid‘ options ’front’, ’back’ and ’layerdefault’ affect the drawing order of the xtics, too.

Examples:

Make tics at 0, 0.5, 1, 1.5, ..., 9.5, 10.

```
set xtics 0,.5,10
```

Make tics at ..., -10, -5, 0, 5, 10, ...

```
set xtics 5
```

Make tics at 1, 100, 1e4, 1e6, 1e8.

```
set logscale x; set xtics 1,100,1e8
```

The explicit ("<label>" <pos> <level>, ...) form allows arbitrary tic positions or non-numeric tic labels. In this form, the tics do not need to be listed in numerical order. Each tic has a position, optionally with a label. Note that the label is a string enclosed by quotes. It may be a constant string, such as "hello", may contain formatting information for converting the position into its label, such as "%3f clients", or may be empty, "". See ‘set format‘ for more information. If no string is given, the default label (numerical) is used.

An explicit tic mark has a third parameter, the "level". The default is level 0, a major tic. A level of 1 generates a minor tic. If the level is specified, then the label must also be supplied.

Examples:

```
set xtics ("low" 0, "medium" 50, "high" 100)
set xtics (1,2,4,8,16,32,64,128,256,512,1024)
set ytics ("bottom" 0, "" 10, "top" 20)
set ytics ("bottom" 0, "" 10 1, "top" 20)
```

In the second example, all tics are labelled. In the third, only the end tics are labelled. In the fourth, the unlabeled tic is a minor tic.

Normally if explicit tics are given, they are used instead of auto-generated tics. Conversely if you specify ‘set xtics auto‘ or the like it will erase any previously specified explicit tics. You can mix explicit and auto- generated tics by using the keyword ‘add‘, which must appear before the tic style being added.

Example:

```
set xtics 0,.5,10
set xtics add ("Pi" 3.14159)
```

This will automatically generate tic marks every 0.5 along x, but will also add an explicit labeled tic mark at pi.

However they are specified, tics will only be plotted when in range.

Format (or omission) of the tic labels is controlled by ‘set format‘, unless the explicit text of a label is included in the ‘set xtics ("<label>")‘ form.

Minor (unlabelled) tics can be added automatically by the Section 2.21.40 [mxtics], page 113 command, or at explicit positions by the ‘set xtics ("" <pos> 1, ...)‘ form.

In case of timeseries data, position values must be given as quoted dates or times according to the format Section 2.21.67 [timefmt], page 147. If the <start>, <incr>, <end> form is used, <start> and <end> must be given according to Section 2.21.67 [timefmt], page 147, but <incr> must be in seconds. Times will be written out according to the format given on ‘set format‘, however.

Examples:

```
set xdata time
set timefmt "%d/%m"
set format x "%b %d"
set xrange ["01/12":"06/12"]
set xtics "01/12", 172800, "05/12"


set xdata time
set timefmt "%d/%m"
set format x "%b %d"
set xrange ["01/12":"06/12"]
set xtics ("01/12", "" "03/12", "05/12")
```

Both of these will produce tics "Dec 1", "Dec 3", and "Dec 5", but in the second example the tic at "Dec 3" will be unlabelled.

2.21.89 xyplane

The Section 2.21.89 [xyplane], page 159 command adjusts the position at which the xy plane is drawn in a 3D plot. The synonym "set ticslevel" is accepted for backwards compatibility.

Syntax:

```
set ticslevel <frac>
set xyplane <frac>
set xyplane at <zvalue>
show xyplane
```

The form ‘set ticslevel <frac>‘ places the xy plane below the range in Z, where the distance from the xy plane to Zmin is given as a fraction of the total range in z. The default value is 0.5. Negative values are permitted, but tic labels on the three axes may overlap.

To place the xy-plane at a position ’pos’ on the z-axis, Section 2.21.64 [ticslevel], page 147 may be set equal to (pos - zmin) / (zmin - zmax). However, this position will change if the z range is changed.

The alternative form ‘set xyplane at <zvalue>‘ fixes the placement of the xy plane at a specific Z value regardless of the current z range. Thus to force the x, y, and z axes to meet at a common origin one would specify ‘set xyplane at 0‘.

See also Section 2.21.74 [view], page 150, and Section 2.21.111 [zeroaxis], page 161.

2.21.90 xzeroaxis

The Section 2.21.90 [xzeroaxis], page 159 command draws a line at y = 0. For details, please see Section 2.21.111 [zeroaxis], page 161.

2.21.91 y2data

The Section 2.21.91 [y2data], page 159 command sets y2 (right-hand) axis data to timeseries (dates/times). Please see Section 2.21.83 [xdata], page 151.

2.21.92 y2dtics

The Section 2.21.92 [y2dtics], page 159 command changes tics on the y2 (right-hand) axis to days of the week. Please see Section 2.21.84 [xdtics], page 152 for details.

2.21.93 y2label

The Section 2.21.93 [y2label], page 159 command sets the label for the y2 (right-hand) axis. Please see Section 2.21.85 [xlabel], page 152.

2.21.94 y2mtics

The Section 2.21.94 [y2mtics], page 159 command changes tics on the y2 (right-hand) axis to months of the year. Please see Section 2.21.86 [xmtics], page 153 for details.

2.21.95 y2range

The Section 2.21.95 [y2range], page 159 command sets the vertical range that will be displayed on the y2 (right-hand) axis. Please see Section 2.21.87 [xrange], page 154 for details.

2.21.96 y2tics

The Section 2.21.96 [y2tics], page 160 command controls major (labelled) tics on the y2 (right-hand) axis. Please see Section 2.21.88 [xtics], page 155 for details.

2.21.97 y2zeroaxis

The Section 2.21.97 [y2zeroaxis], page 160 command draws a line at the origin of the y2 (right-hand) axis (x2 = 0). For details, please see Section 2.21.111 [zeroaxis], page 161.

2.21.98 ydata

The Section 2.21.98 [ydata], page 160 commands sets y-axis data to timeseries (dates/times). Please see Section 2.21.83 [xdata], page 151.

2.21.99 ydtics

The Section 2.21.99 [ydtics], page 160 command changes tics on the y axis to days of the week. Please see Section 2.21.84 [xdtics], page 152 for details.

2.21.100 ylabel

This command sets the label for the y axis. Please see Section 2.21.85 [xlabel], page 152.

2.21.101 ymtics

The Section 2.21.101 [ymtics], page 160 command changes tics on the y axis to months of the year. Please see Section 2.21.86 [xmtics], page 153 for details.

2.21.102 yrange

The Section 2.21.102 [yrange], page 160 command sets the vertical range that will be displayed on the y axis. Please see Section 2.21.87 [xrange], page 154 for details.

2.21.103 ytics

The Section 2.21.103 [ytics], page 160 command controls major (labelled) tics on the y axis. Please see Section 2.21.88 [xtics], page 155 for details.

2.21.104 yzeroaxis

The Section 2.21.104 [yzeroaxis], page 160 command draws a line at x = 0. For details, please see Section 2.21.111 [zeroaxis], page 161.

2.21.105 zdata

The Section 2.21.105 [zdata], page 160 command sets zaxis data to timeseries (dates/times). Please see Section 2.21.83 [xdata], page 151.

2.21.106 zdtics

The Section 2.21.106 [zdtics], page 160 command changes tics on the z axis to days of the week. Please see Section 2.21.84 [xdtics], page 152 for details.

2.21.107 zzeroaxis

The Section 2.21.107 [zzeroaxis], page 161 command draws a line through (x=0,y=0). This has no effect on 2D plots, including splot with ‘set view map‘. For details, please see Section 2.21.111 [zeroaxis], page 161 and Section 2.21.89 [xyplane], page 159.

2.21.108 cbdata

Set color box axis data to timeseries (dates/times). Please see Section 2.21.83 [xdata], page 151.

2.21.109 cbdtics

The Section 2.21.109 [cbdtics], page 161 command changes tics on the color box axis to days of the week. Please see Section 2.21.84 [xdtics], page 152 for details.

2.21.110 zero

The ‘zero‘ value is the default threshold for values approaching 0.0.

Syntax:

```
set zero <expression>
show zero
```

‘gnuplot‘ will not plot a point if its imaginary part is greater in magnitude than the ‘zero‘ threshold. This threshold is also used in various other parts of ‘gnuplot‘ as a (crude) numerical-error threshold. The default ‘zero‘ value is 1e-8. ‘zero‘ values larger than 1e-3 (the reciprocal of the number of pixels in a typical bitmap display) should probably be avoided, but it is not unreasonable to set ‘zero‘ to 0.0.

2.21.111 zeroaxis

The x axis may be drawn by Section 2.21.90 [xzeroaxis], page 159 and removed by Section 2.21.90 [xzeroaxis], page 159. Similar commands behave similarly for the y, x2, y2, and z axes.

Syntax:

```
set {x|x2|y|y2|z}zeroaxis { {linestyle | ls <line_style>}
                           | { linetype | lt <line_type>}
                             { linewidth | lw <line_width>}}
unset {x|x2|y|y2|z}zeroaxis
show {x|y|z}zeroaxis
```

By default, these options are off. The selected zero axis is drawn with a line of type <line_type> and width <line_width> (if supported by the terminal driver currently in use), or a user-defined style <line_style>.

If no linetype is specified, any zero axes selected will be drawn using the axis linetype (linetype 0).

Section 2.21.111 [zeroaxis], page 161 is equivalent to Section 2.21.104 [yzeroaxis], page 160. Note that the z-axis must be set separately using Section 2.21.107 [zzeroaxis], page 161.

Examples:

To simply have the y=0 axis drawn visibly:

```
set xzeroaxis
```

If you want a thick line in a different color or pattern, instead:

```
set xzeroaxis linetype 3 linewidth 2.5
```

2.21.112 zlabel

This command sets the label for the z axis. Please see Section 2.21.85 [xlabel], page 152.

2.21.113 zmtics

The Section 2.21.113 [zmtics], page 162 command changes tics on the z axis to months of the year. Please see Section 2.21.86 [xmtics], page 153 for details.

2.21.114 zrange

The Section 2.21.114 [zrange], page 162 command sets the range that will be displayed on the z axis. The zrange is used only by 'splot' and is ignored by 'plot'. Please see Section 2.21.87 [xrange], page 154 for details.

2.21.115 ztics

The Section 2.21.115 [ztics], page 162 command controls major (labelled) tics on the z axis. Please see Section 2.21.88 [xtics], page 155 for details.

2.21.116 cblabel

This command sets the label for the color box axis. Please see Section 2.21.85 [xlabel], page 152.

2.21.117 cbmtics

The Section 2.21.117 [cbmtics], page 162 command changes tics on the color box axis to months of the year. Please see Section 2.21.86 [xmtics], page 153 for details.

2.21.118 cbrange

The Section 2.21.118 [cbrange], page 162 command sets the range of values which are colored according to the current Section 2.21.50 [palette], page 120 by styles pm3d, 'with image' and Section 2.21.50 [palette], page 120. Values outside of the color range use color of the nearest extreme.

If the cb-axis is autoscaled in 'splot', then the range is taken from Section 2.21.114 [zrange], page 162. Points drawn in 'splot ... pm3d|palette' can be filtered by using different Section 2.21.114 [zrange], page 162 and Section 2.21.118 [cbrange], page 162.

Please see Section 2.21.87 [xrange], page 154 for details on Section 2.21.118 [cbrange], page 162 syntax. See also Section 2.21.50 [palette], page 120 and 'set colorbox'.

2.21.119 cbtics

The Section 2.21.119 [cbtics], page 162 command controls major (labelled) tics on the color box axis. Please see Section 2.21.88 [xtics], page 155 for details.

2.22 shell

The Section 2.22 [shell], page 163 command spawns an interactive shell. To return to ‘gnuplot‘, type ‘logout‘ if using VMS, Section 2.4 [exit], page 36 or the END-OF-FILE character if using Unix, ‘endcli‘ if using AmigaOS, or Section 2.4 [exit], page 36 if using MS-DOS or OS/2.

There are two ways of spawning a shell command: using ‘system‘ command or via ‘¡ ($ if using VMS). The former command takes a string as a parameter and thus it can be used anywhere among other gnuplot commands, while the latter syntax requires to be the only command on the line. Control will return immediately to ‘gnuplot‘ after this command is executed. For example, in AmigaOS, MS-DOS or OS/2,

```
! dir
```

or

```
system "dir"
```

prints a directory listing and then returns to ‘gnuplot‘.

Other examples of the former syntax:

```
system "date"; set time; plot "a.dat"
print=1; if (print) replot; set out; system "lpr x.ps"
```

On an Atari, the ‘¡ command first checks whether a shell is already loaded and uses it, if available. This is practical if ‘gnuplot‘ is run from ‘gulam‘, for example.

2.23 splot

‘splot‘ is the command for drawing 3-d plots (well, actually projections on a 2-d surface, but you knew that). It can create a plot from functions or a data file in a manner very similar to the ‘plot‘ command.

See ‘plot‘ for features common to the ‘plot‘ command; only differences are discussed in detail here. Note specifically ‘plot‘’s ‘axes‘ option is not available for ‘splot‘.

Syntax:

```
splot {<ranges>}
      <function> | "<datafile>" {datafile-modifiers}}
      {<title-spec>} {with <style>}
      {, {definitions,} <function> ...}
```

where either a <function> or the name of a data file enclosed in quotes is supplied. The function can be a mathematical expression, or a triple of mathematical expressions in parametric mode.

By default ‘splot‘ draws the xy plane completely below the plotted data. The offset between the lowest ztic and the xy plane can be changed by Section 2.21.64 [ticslevel], page 147. The orientation of a ‘splot‘ projection is controlled by Section 2.21.74 [view], page 150. See Section 2.21.74 [view], page 150 and Section 2.21.64 [ticslevel], page 147 for more information.

The syntax for setting ranges on the ‘splot‘ command is the same as for ‘plot‘. In non-parametric mode, the order in which ranges must be given is Section 2.21.87 [xrange], page 154, Section 2.21.102 [yrange], page 160, and Section 2.21.114 [zrange], page 162. In parametric mode, the order is Section 2.21.71 [urange], page 149, Section 2.21.75 [vrange], page 150, Section 2.21.87 [xrange], page 154, Section 2.21.102 [yrange], page 160, and Section 2.21.114 [zrange], page 162.

The ‘title‘ option is the same as in ‘plot‘. The operation of Section 2.12.7 [with], page 68 is also the same as in ‘plot‘, except that the plotting styles available to ‘splot‘ are limited to ‘lines‘, ‘points‘, ‘linespoints‘, ‘dots‘, and ‘impulses‘; the error-bar capabilities of ‘plot‘ are not available for ‘splot‘.

The Section 2.21.14 [datafile], page 86 options have more differences.

See also ‘show plot‘.

2.23.1 data-file

As for ‘plot‘, discrete data contained in a file can be displayed by specifying the name of the data file, enclosed in quotes, on the ‘splot‘ command line.

Syntax:

```
splot '<file_name>' {binary <binary list>}
                    {matrix}
                    {index <index list>}
                    {every <every list>}
                    {using <using list>}
```

The special filenames ‘""‘ and ‘"-"‘ are permitted, as in ‘plot‘.

In brief, ‘binary‘ and ‘matrix‘ indicate that the data are in a special form, Section 2.12.1.5 [index], page 57 selects which data sets in a multi-data-set file are to be plotted, Section 2.12.1.3 [every], page 56 specifies which datalines (subsets) within a single data set are to be plotted, and Section 2.12.1.9 [using], page 61 determines how the columns within a single record are to be interpreted.

The options Section 2.12.1.5 [index], page 57 and Section 2.12.1.3 [every], page 56 behave the same way as with ‘plot‘; Section 2.12.1.9 [using], page 61 does so also, except that the Section 2.12.1.9 [using], page 61 list must provide three entries instead of two.

The ‘plot‘ options Section 2.12.1.8 [thru], page 60 and Section 2.12.1.6 [smooth], page 57 are not available for ‘splot‘, but Section 2.21.10 [cntrparam], page 83 and Section 2.21.16 [dgrid3d], page 90 provide limited smoothing capabilities.

Data file organization is essentially the same as for ‘plot‘, except that each point is an (x,y,z) triple. If only a single value is provided, it will be used for z, the datablock number will be used for y, and the index of the data point in the datablock will be used for x. If two or four values are provided, ‘gnuplot‘ uses the last value for calculating the color in pm3d plots. Three values are interpreted as an (x,y,z) triple. Additional values are generally used as errors, which can be used by ‘fit‘.

Single blank records separate datablocks in a ‘splot‘ datafile; ‘splot‘ treats datablocks as the equivalent of function y-isolines. No line will join points separated by a blank record. If all datablocks contain the same number of points, ‘gnuplot‘ will draw cross-isolines between datablocks, connecting corresponding points. This is termed "grid data", and is required for drawing a surface, for contouring (Section 2.21.12 [contour], page 86) and hidden-line removal (Section 2.21.25 [hidden3d], page 97). See also Section 2.23.2 [grid_data], page 167.

It is no longer necessary to specify ‘parametric‘ mode for three-column ‘splot‘s.

2.23.1.1 binary matrix

Gnuplot can read matrix binary files by use of the option ‘binary‘ appearing without keyword qualifications unique to general binary, i.e., ‘array‘, ‘record‘, ‘format‘, or ‘filetype‘. Other general binary keywords for translation should also apply to matrix binary. (See ‘binary general‘ for more details.)

In previous versions, ‘gnuplot‘ dynamically detected binary data files. It is now necessary to specify the keyword ‘binary‘ directly after the filename.

Single precision floats are stored in a binary file as follows:

```
<N+1>  <y0>   <y1>   <y2>  ...  <yN>
 <x0> <z0,0> <z0,1> <z0,2> ... <z0,N>
 <x1> <z1,0> <z1,1> <z1,2> ... <z1,N>
  :      :      :      :   ...    :
```

which are converted into triplets:

```
<x0> <y0> <z0,0>
<x0> <y1> <z0,1>
<x0> <y2> <z0,2>
 :    :     :
<x0> <yN> <z0,N>

<x1> <y0> <z1,0>
<x1> <y1> <z1,1>
 :    :     :
```

These triplets are then converted into ‘gnuplot‘ iso-curves and then ‘gnuplot‘ proceeds in the usual manner to do the rest of the plotting.

A collection of matrix and vector manipulation routines (in C) is provided in ‘binary.c‘. The routine to write binary data is

```
int fwrite_matrix(file,m,nrl,nrl,ncl,nch,row_title,column_title)
```

An example of using these routines is provided in the file ‘bf_test.c‘, which generates binary files for the demo file ‘demo/binary.dem‘.

The Section 2.12.1.5 [index], page 57 keyword is not supported, since the file format allows only one surface per file. The Section 2.12.1.3 [every], page 56 and Section 2.12.1.9 [using], page 61 filters are supported. Section 2.12.1.9 [using], page 61 operates as if the data were read in the above triplet form.

See also ‘binary general‘ and

Binary File Splot Demo.

2.23.1.2 example datafile

A simple example of plotting a 3-d data file is

```
splot 'datafile.dat'
```

where the file "datafile.dat" might contain:

```
# The valley of the Gnu.
   0 0 10
   0 1 10
   0 2 10

   1 0 10
   1 1 5
   1 2 10

   2 0 10
   2 1 1
   2 2 10

   3 0 10
   3 1 0
   3 2 10
```

Note that "datafile.dat" defines a 4 by 3 grid (4 rows of 3 points each). Rows (datablocks) are separated by blank records.

Note also that the x value is held constant within each dataline. If you instead keep y constant, and plot with hidden-line removal enabled, you will find that the surface is drawn 'inside-out'.

Actually for grid data it is not necessary to keep the x values constant within a datablock, nor is it necessary to keep the same sequence of y values. 'gnuplot' requires only that the number of points be the same for each datablock. However since the surface mesh, from which contours are derived, connects sequentially corresponding points, the effect of an irregular grid on a surface plot is unpredictable and should be examined on a case-by-case basis.

2.23.1.3 matrix_ascii

The 'matrix' keyword (without a sequent 'binary' keyword) in

```
{s}plot 'a.dat' matrix
```

indicates that data are stored in an ascii numbers matrix format.

The z-values are read in a row at a time, i. e.,

```
z11 z12 z13 z14 ...
z21 z22 z23 z24 ...
z31 z32 z33 z34 ...
```

and so forth.

In 3D, the x- and y-indices of the matrix surface plot correspond to column and row indices of the matrix, respectively, being enumerated from 0. You can rescale or transform the axes as usual for a data file with three columns by means of x=$1, y=$2, z=$3. For example

```
splot 'a.dat' matrix using (1+$1/100):(1+$2*10):3
```

A blank line or comment line ends the matrix, and starts a new surface mesh. You can select among the meshes inside a file by the Section 2.12.1.5 [index], page 57 option to the 'splot' command, as usual.

See 'matrix' for examples of plotting rows and columns of the matrix in a 2D plot.

2.23.1.4 matrix

Datafile can be in an ascii or binary matrix format. The 'matrix' flag indicates that the file is ascii, the 'binary' or 'matrix binary' stands for a binary format. For details, see 'matrix ascii' and 'matrix binary'.

Basic usage in 'splot':

```
splot 'a.dat' matrix
splot 'a.gpbin' {matrix} binary
```

Advanced usage in 'splot':

```
splot 'a.dat' matrix using 1:2:3
splot 'a.gpbin' {matrix} binary using 1:2:3
```

allows to transform the axes coordinates and the z-data independently.

Usage in 'plot':

```
plot `a.dat` matrix
plot `a.dat` matrix using 1:3
plot 'a.gpbin' {matrix} binary using 1:3
```

will plot rows of the matrix, while using 2:3 will plot matrix columns, and using 1:2 the point coordinates (rather useless). Applying the Section 2.12.1.3 [every], page 56 option you can specify explicit rows and columns.

Example – rescale axes of a matrix in an ascii file:

```
splot `a.dat` matrix using (1+$1):(1+$2*10):3
```

Example – plot the 3rd row of a matrix in an ascii file:

```
plot 'a.dat' matrix using 1:3 every 1:999:1:2
```

(rows are enumerated from 0, thus 2 instead of 3).

2.23.2 grid_data

The 3D routines are designed for points in a grid format, with one sample, datapoint, at each mesh intersection; the datapoints may originate from either evaluating a function, see Section 2.21.27 [isosamples], page 100, or reading a datafile, see Section 2.21.14 [datafile], page 86. The term "isoline" is applied to the mesh lines for both functions and data. Note that the mesh need not be rectangular in x and y, as it may be parameterized in u and v, see Section 2.21.27 [isosamples], page 100.

However, ‘gnuplot‘ does not require that format. In the case of functions, ’samples’ need not be equal to ’isosamples’, i.e., not every x-isoline sample need intersect a y-isoline. In the case of data files, if there are an equal number of scattered data points in each datablock, then "isolines" will connect the points in a datablock, and "cross-isolines" will connect the corresponding points in each datablock to generate a "surface". In either case, contour and hidden3d modes may give different plots than if the points were in the intended format. Scattered data can be converted to a {different} grid format with Section 2.21.16 [dgrid3d], page 90.

The contour code tests for z intensity along a line between a point on a y-isoline and the corresponding point in the next y-isoline. Thus a ‘splot‘ contour of a surface with samples on the x-isolines that do not coincide with a y-isoline intersection will ignore such samples. Try:

```
set xrange [-pi/2:pi/2]; set yrange [-pi/2:pi/2]
set style function lp
set contour
set isosamples 10,10; set samples 10,10;
splot cos(x)*cos(y)
set samples 4,10; replot
set samples 10,4; replot
```

2.23.3 splot_overview

‘splot‘ can display a surface as a collection of points, or by connecting those points. As with ‘plot‘, the points may be read from a data file or result from evaluation of a function at specified intervals, see Section 2.21.27 [isosamples], page 100. The surface may be approximated by connecting the points with straight line segments, see Section 2.21.59 [surface], page 144, in which case the surface can be made opaque with ‘set hidden3d.‘ The orientation from which the 3d surface is viewed can be changed with Section 2.21.74 [view], page 150.

Additionally, for points in a grid format, ‘splot‘ can interpolate points having a common amplitude (see Section 2.21.12 [contour], page 86) and can then connect those new points to display contour lines, either directly with straight-line segments or smoothed lines (see Section 2.21.10 [cntrparam], page 83). Functions are already evaluated in a grid format, determined by Section 2.21.27 [isosamples], page 100 and Section 2.21.56 [samples], page 128, while file data must either be in a grid format, as described in Section 2.23.1 [data-file], page 164, or be used to generate a grid (see Section 2.21.16 [dgrid3d], page 90).

Contour lines may be displayed either on the surface or projected onto the base. The base projections of the contour lines may be written to a file, and then read with ‘plot‘, to take advantage of ‘plot‘’s additional formatting capabilities.

2.24 system

‘system‘ spawns shell to execute a command. Please type Section 2.22 [shell], page 163 for more details.

2.25 test

This command graphically tests or presents terminal and palette capabilities.

Syntax:

```
test {terminal | palette [rgb|rbg|grb|gbr|brg|bgr]}
```

Section 2.25 [test], page 168 or ‘test terminal‘ creates a display of line and point styles and other useful things appropriate for and supported by the ‘terminal‘ you are just using.

Section 2.21.50 [palette], page 120 draws graphically profiles R(z),G(z),B(z), where 0<=z<=1, as calculated by the current color Section 2.21.50 [palette], page 120. In other words, it is a beautiful plot you would have to do yourself with the result of ‘show palette palette 256 float‘. The optional parameter, a permutation of letters rgb, determines the sequence of r,g,b profiles drawn one after the other — try this yourself for ‘set palette gray‘. The default sequence is rgb.

2.26 unset

Options set using the ‘set‘ command may be returned to their default state by issuing the corresponding Section 2.26 [unset], page 169 command.

Example:

```
set xtics mirror rotate by -45 0,10,100
...
unset xtics
```

2.27 update

This command writes the current values of the fit parameters into the given file, formatted as an initial-value file (as described in the ‘fit‘section). This is useful for saving the current values for later use or for restarting a converged or stopped fit.

Syntax:

```
update <filename> {<filename>}
```

If a second filename is supplied, the updated values are written to this file, and the original parameter file is left unmodified.

Otherwise, if the file already exists, ‘gnuplot‘ first renames it by appending ‘.old‘ and then opens a new file. That is, "‘update ’fred’‘" behaves the same as "‘!rename fred fred.old; update ’fred.old’ ’fred’‘". [On DOS and other systems that use the twelve-character "filename.ext" naming convention, "ext" will be "‘old‘" and "filename" will be related (hopefully recognizably) to the initial name. Renaming is not done at all on VMS systems, since they use file-versioning.]

Please see ‘fit‘ for more information.

3 Terminal types

3.1 terminal

Gnuplot supports a large number of output formats. These are selected by choosing an appropriate terminal type, possibly with additional modifying options. See ‘set terminal‘.

This document may describe terminal types that are not available to you because they were not configured or installed on your system. To see a list of terminals available on a particular gnuplot installation, type ’set terminal’ with no modifiers. @c <3 – all terminal stuff is pulled from the .trm files

3.1.0.1 aed767

The ‘aed512‘ and ‘aed767‘ terminal drivers support AED graphics terminals. The two drivers differ only in their horizontal ranges, which are 512 and 768 pixels, respectively. Their vertical range is 575 pixels. There are no options for these drivers."

3.1.0.2 aifm

Several options may be set in ‘aifm‘—the Adobe Illustrator 3.0+ driver.

Syntax:

```
set terminal aifm {<color>} {"<fontname>"} {<fontsize>}
```

<color> is either ‘color‘ or ‘monochrome‘; "<fontname>" is the name of a valid PostScript font; <fontsize> is the size of the font in PostScript points, before scaling by the Section 2.21.57 [size], page 129 command. Selecting ‘default‘ sets all options to their default values: ‘monochrome‘, "Times-Roman", and 14pt.

Since AI does not really support multiple pages, multiple graphs will be drawn directly on top of one another. However, each graph will be grouped individually, making it easy to separate them inside AI (just pick them up and move them).

Examples:

```
set term aifm
set term aifm 22
set size 0.7,1.4; set term aifm color "Times-Roman" 14"
```

3.1.0.3 amiga

The ‘amiga‘ terminal, for Commodore Amiga computers, allows the user to plot either to a screen (default), or, if Kickstart 3.0 or higher is installed, to a window on the current public screen. The font and its size can also be selected.

Syntax:

```
set terminal amiga {screen | window} {"<fontname>"} {<fontsize>}
```

The default font is 8-point "topaz".

The screen option uses a virtual screen, so it is possible that the graph will be larger than the screen."

3.1.0.4 apollo

The ‘apollo‘ terminal driver supports the Apollo Graphics Primitive Resource with rescaling after window resizing. It has no options.

If a fixed-size window is desired, the ‘gpr‘ terminal may be used instead."

3.1.0.5 aqua

This terminal relies on AquaTerm.app for display on Mac OS X.

Syntax:

```
set terminal aqua {<n>} {title "<wintitle>"} {size <x> <y>}
                  {fname "<fontface>"} {fsize <fontsize>}
                  {{no}enhanced}
```

where <n> is the number of the window to draw in (default is 0), <wintitle> is the name shown in the title bar (default "Figure <n>"), <x> <y> is the size of the plot (default is 846x594 pt = 11.75x8.25 in).

Use <fontface> to specify the font to use (default is "Times-Roman"), <fontzise> sets the font size (default is 14.0 pt).

The aqua terminal support enhanced text mode (see ‘enhanced‘), except for overprint. Font support is limited to the fonts available on the system. Character encoding can be selected by Section 2.21.18 [encoding], page 91 and currently supports iso_latin_1, iso_latin2, and cp1250 with iso_latin_1 being the dafault."

3.1.0.6 atari ST (via AES)

The ‘atari‘ terminal has options to set the character size and the screen colors.

Syntax:

```
set terminal atari {<fontsize>} {<col0> <col1> ... <col15>}
```

The character size must appear if any colors are to be specified. Each of the (up to 16) colors is given as a three-digit hex number, where the digits represent RED, GREEN and BLUE (in that order). The range of 0–15 is scaled to whatever color range the screen actually has. On a normal ST screen, odd and even intensities are the same.

Examples:

```
      set terminal atari 4    # use small (6x6) font
      set termi-
nal atari 6 0  # set monochrome screen to white on black
      set terminal atari 13 0 fff f00 f0 f ff f0f
                 # set first seven colors to black, white, red, green,
                 # blue, cyan, and purple and use large font (8x16).
```

Additionally, if an environment variable GNUCOLORS exists, its contents are interpreted as an options string, but an explicit terminal option takes precedence."

3.1.0.7 atari ST (via VDI)

The ‘vdi‘ terminal is the same as the ‘atari‘ terminal, except that it sends output to the screen via the VDI and not into AES-Windows.

The ‘vdi‘ terminal has options to set the character size and the screen colors.

Syntax:

```
set terminal vdi {<fontsize>} {<col0> <col1> ... <col15>}
```

The character size must appear if any colors are to be specified. Each of the (up to 16) colors is given as a three-digit hex number, where the digits represent RED, GREEN and BLUE (in that order). The range of 0–15 is scaled to whatever color range the screen actually has. On a normal ST screen, odd and even intensities are the same.

Examples:

```
set terminal vdi 4    # use small (6x6) font
set terminal vdi 6 0  # set monochrome screen to white on black
set terminal vdi 13 0 fff f00 f0 f ff f0f
           # set first seven colors to black, white, red, green,
           # blue, cyan, and purple and use large font (8x16).
```

Additionally, if an environment variable GNUCOLORS exists, its contents are interpreted as an options string, but an explicit terminal option takes precedence."

3.1.0.8 be

‘gnuplot‘ provides the ‘be‘ terminal type for use with X servers. This terminal type is set automatically at startup if the ‘DISPLAY‘ environment variable is set, if the ‘TERM‘ environment variable is set to ‘xterm‘, or if the ‘-display‘ command line option is used.

Syntax:

```
set terminal be {reset} {<n>}
```

Multiple plot windows are supported: ‘set terminal be <n>‘ directs the output to plot window number n. If n>0, the terminal number will be appended to the window title and the icon will be labeled ‘gplt <n>‘. The active window may distinguished by a change in cursor (from default to crosshair.)

Plot windows remain open even when the ‘gnuplot‘ driver is changed to a different device. A plot window can be closed by pressing the letter q while that window has input focus, or by choosing ‘close‘ from a window manager menu. All plot windows can be closed by specifying Section 2.19 [reset], page 73, which actually terminates the subprocess which maintains the windows (unless ‘-persist‘ was specified).

Plot windows will automatically be closed at the end of the session unless the ‘-persist‘ option was given.

The size or aspect ratio of a plot may be changed by resizing the ‘gnuplot‘ window.

Linewidths and pointsizes may be changed from within ‘gnuplot‘ with ‘set linestyle‘.

For terminal type ‘be‘, ‘gnuplot‘ accepts (when initialized) the standard X Toolkit options and resources such as geometry, font, and name from the command line arguments or a configuration file. See the X(1) man page (or its equivalent) for a description of such options.

A number of other ‘gnuplot‘ options are available for the ‘be‘ terminal. These may be specified either as command-line options when ‘gnuplot‘ is invoked or as resources in the configuration file ".Xdefaults". They are set upon initialization and cannot be altered during a ‘gnuplot‘ session.

— COMMAND-LINE_OPTIONS —

In addition to the X Toolkit options, the following options may be specified on the command line when starting ‘gnuplot‘ or as resources in your ".Xdefaults" file:

```
 ‘-mono‘        forces monochrome rendering on color displays.
 ‘-gray‘        requests grayscale render-
ing on grayscale or color displays.
                       (Grayscale displays receive monochrome render-
ing by default.)
 ‘-clear‘   requests that the window be cleared momentarily before a
                       new plot is displayed.
 ‘-raise‘   raises plot window after each plot
 ‘-noraise‘ does not raise plot window after each plot
 ‘-persist‘ plots windows survive after main gnuplot program exits
```

The options are shown above in their command-line syntax. When entered as resources in ".Xdefaults", they require a different syntax.

Example:

```
gnuplot*gray: on
```

‘gnuplot‘ also provides a command line option (‘-pointsize <v>‘) and a resource, ‘gnuplot*pointsize: <v>‘, to control the size of points plotted with the ‘points‘ plotting style. The value ‘v‘ is a real number (greater than 0 and less than or equal to ten) used as a scaling factor for point sizes. For example, ‘-pointsize 2‘ uses points twice the default size, and ‘-pointsize 0.5‘ uses points half the normal size.

— MONOCHROME_OPTIONS —

For monochrome displays, ‘gnuplot‘ does not honor foreground or background colors. The default is black-on-white. ‘-rv‘ or ‘gnuplot*reverseVideo: on‘ requests white-on-black.

— COLOR_RESOURCES —

For color displays, ‘gnuplot‘ honors the following resources (shown here with their default values) or the greyscale resources. The values may be color names as listed in the BE rgb.txt file on your system, hexadecimal RGB color specifications (see BE documentation), or a color name followed by a comma and an ‘intensity‘ value from 0 to 1. For example, ‘blue, 0.5‘ means a half intensity blue.

```
gnuplot*background:  white
gnuplot*textColor:   black
gnuplot*borderColor: black
gnuplot*axisColor:   black
gnuplot*line1Color:  red
```

```
gnuplot*line2Color:  green
gnuplot*line3Color:  blue
gnuplot*line4Color:  magenta
gnuplot*line5Color:  cyan
gnuplot*line6Color:  sienna
gnuplot*line7Color:  orange
gnuplot*line8Color:  coral
```

The command-line syntax for these is, for example,

Example:

```
gnuplot -background coral
```

— GRAYSCALE_RESOURCES —

When '-gray' is selected, 'gnuplot' honors the following resources for grayscale or color displays (shown here with their default values). Note that the default background is black.

```
gnuplot*background: black
gnuplot*textGray:   white
gnuplot*borderGray: gray50
gnuplot*axisGray:   gray50
gnuplot*line1Gray:  gray100
gnuplot*line2Gray:  gray60
gnuplot*line3Gray:  gray80
gnuplot*line4Gray:  gray40
gnuplot*line5Gray:  gray90
gnuplot*line6Gray:  gray50
gnuplot*line7Gray:  gray70
gnuplot*line8Gray:  gray30
```

— LINE_RESOURCES —

'gnuplot' honors the following resources for setting the width (in pixels) of plot lines (shown here with their default values.) 0 or 1 means a minimal width line of 1 pixel width. A value of 2 or 3 may improve the appearance of some plots.

```
gnuplot*borderWidth: 2
gnuplot*axisWidth:   0
gnuplot*line1Width:  0
gnuplot*line2Width:  0
gnuplot*line3Width:  0
gnuplot*line4Width:  0
gnuplot*line5Width:  0
gnuplot*line6Width:  0
gnuplot*line7Width:  0
gnuplot*line8Width:  0
```

'gnuplot' honors the following resources for setting the dash style used for plotting lines. 0 means a solid line. A two-digit number 'jk' ('j' and 'k' are >= 1 and <= 9) means a dashed line with a repeated pattern of ‘j‘ pixels on followed by ‘k‘ pixels off. For example, ’16’ is a "dotted" line with one pixel on followed by six pixels off. More elaborate on/off patterns can be specified with a four-digit value. For example, ’4441’ is four on, four off, four on, one off. The default values shown below are for monochrome displays or monochrome rendering on color or grayscale displays. For color displays, the default for each is 0 (solid line) except for ‘axisDashes‘ which defaults to a ’16’ dotted line.

```
gnuplot*borderDashes:   0
gnuplot*axisDashes:         16
gnuplot*line1Dashes:         0
gnuplot*line2Dashes:   42
gnuplot*line3Dashes:   13
gnuplot*line4Dashes:   44
gnuplot*line5Dashes:   15
gnuplot*line6Dashes: 4441
gnuplot*line7Dashes:   42
gnuplot*line8Dashes:   13
```

3.1.0.9 cgi

The ‘cgi‘ and ‘hcgi‘ terminal drivers support SCO CGI drivers. ‘hcgi‘ is for printers; the environment variable CGIPRNT must be set. ‘cgi‘ may be used for either a display or hardcopy; if the environment variable CGIDISP is set, then that display is used. Otherwise CGIPRNT is used.

These terminals have no options."

3.1.0.10 cgm

The ‘cgm‘ terminal generates a Computer Graphics Metafile, Version 1. This file format is a subset of the ANSI X3.122-1986 standard entitled "Computer Graphics - Metafile for the Storage and Transfer of Picture Description Information". Several options may be set in ‘cgm‘.

Syntax:

```
        set terminal cgm {<mode>} {<color>} {<rota-
tion>} {solid | dashed}
                         {width <plot_width>} {linewidth <line_width>}
                         {"<font>"} {<fontsize>}
                         {<color0> <color1> <color2> ...}
```

where <mode> is ‘landscape‘, ‘portrait‘, or ‘default‘; <color> is either ‘color‘ or ‘monochrome‘; <rotation> is either ‘rotate‘ or ‘norotate‘; ‘solid‘ draws all curves with solid lines, overriding any dashed patterns; <plot_width> is the assumed width of the plot in points; <line_width> is the line width in points (default 1); <font> is the name of a font; and <fontsize> is the size of the font in points (default 12).

By default, ‘cgm‘ uses rotated text for the Y axis label.

The first six options can be in any order. Selecting ‘default‘ sets all options to their default values.

Each color must be of the form 'xrrggbb', where x is the literal character 'x' and 'rrggbb' are the red, green and blue components in hex. For example, 'x00ff00' is green. The background color is set first, then the plotting colors.

Examples:

```
set terminal cgm landscape color rotate dashed width 432 \\
               linewidth 1  'Helvetica Bold' 12       # defaults
set terminal cgm linewidth 2  14  # wider lines & larger font
set terminal cgm portrait "Times Italic" 12
set terminal cgm color solid      # no pesky dashes!
```

— FONT —

The first part of a Computer Graphics Metafile, the metafile description, includes a font table. In the picture body, a font is designated by an index into this table. By default, this terminal generates a table with the following 35 fonts, plus six more with 'italic' replaced by 'oblique', or vice-versa (since at least the Microsoft Office and Corel Draw CGM import filters treat 'italic' and 'oblique' as equivalent):

```
Helvetica
Helvetica Bold
Helvetica Oblique
Helvetica Bold Oblique
Times Roman
Times Bold
Times Italic
Times Bold Italic
Courier
Courier Bold
Courier Oblique
Courier Bold Oblique
Symbol
Hershey/Cartographic_Roman
Hershey/Cartographic_Greek
Hershey/Simplex_Roman
Hershey/Simplex_Greek
Hershey/Simplex_Script
Hershey/Complex_Roman
Hershey/Complex_Greek
Hershey/Complex_Script
Hershey/Complex_Italic
Hershey/Complex_Cyrillic
Hershey/Duplex_Roman
Hershey/Triplex_Roman
Hershey/Triplex_Italic
Hershey/Gothic_German
Hershey/Gothic_English
Hershey/Gothic_Italian
Hershey/Symbol_Set_1
```

```
Hershey/Symbol_Set_2
Hershey/Symbol_Math
ZapfDingbats
Script
15
```

The first thirteen of these fonts are required for WebCGM. The Microsoft Office CGM import filter implements the 13 standard fonts listed above, and also 'ZapfDingbats' and 'Script'. However, the script font may only be accessed under the name '15'. For more on Microsoft import filter font substitutions, check its help file which you may find here:

```
C:\\Program Files\\Microsoft Office\\Office\\Cgmimp32.hlp
```

and/or its configuration file, which you may find here:

```
C:\\Program Files\\Common Files\\Microsoft Shared\\Grphflt\\Cgmimp32.cfg
```

In the 'set term' command, you may specify a font name which does not appear in the default font table. In that case, a new font table is constructed with the specified font as its first entry. You must ensure that the spelling, capitalization, and spacing of the name are appropriate for the application that will read the CGM file. (Gnuplot and any MIL-D-28003A compliant application ignore case in font names.) If you need to add several new fonts, use several 'set term' commands.

Example:

```
set terminal cgm 'Old English'
set terminal cgm 'Tengwar'
set terminal cgm 'Arabic'
set output 'myfile.cgm'
plot ...
set output
```

You cannot introduce a new font in a Section 2.21.29 [label], page 104 command.

— FONTSIZE —

Fonts are scaled assuming the page is 6 inches wide. If the Section 2.21.57 [size], page 129 command is used to change the aspect ratio of the page or the CGM file is converted to a different width, the resulting font sizes will be scaled up or down accordingly. To change the assumed width, use the 'width' option.

— LINEWIDTH —

The 'linewidth' option sets the width of lines in pt. The default width is 1 pt. Scaling is affected by the actual width of the page, as discussed under the 'fontsize' and 'width' options.

— ROTATE —

The 'norotate' option may be used to disable text rotation. For example, the CGM input filter for Word for Windows 6.0c can accept rotated text, but the DRAW editor within Word cannot. If you edit a graph (for example, to label a curve), all rotated text is restored to horizontal. The Y axis label will then extend beyond the clip boundary. With 'norotate', the Y axis label starts in a less attractive location, but the page can be edited without damage. The ‘rotate‘ option confirms the default behavior.

— SOLID —

The ‘solid‘ option may be used to disable dashed line styles in the plots. This is useful when color is enabled and the dashing of the lines detracts from the appearance of the plot. The ‘dashed‘ option confirms the default behavior, which gives a different dash pattern to each curve.

— SIZE —

Default size of a CGM plot is 32599 units wide and 23457 units high for landscape, or 23457 units wide by 32599 units high for portrait.

— WIDTH —

All distances in the CGM file are in abstract units. The application that reads the file determines the size of the final plot. By default, the width of the final plot is assumed to be 6 inches (15.24 cm). This distance is used to calculate the correct font size, and may be changed with the ‘width‘ option. The keyword should be followed by the width in points. (Here, a point is 1/72 inch, as in PostScript. This unit is known as a "big point" in TeX.) Gnuplot ‘expressions‘ can be used to convert from other units.

Example:

```
set terminal cgm width 432            # default
set terminal cgm width 6*72           # same as above
set terminal cgm width 10/2.54*72     # 10 cm wide
```

— NOFONTLIST —

The default font table includes the fonts recommended for WebCGM, which are compatible with the Computer Graphics Metafile input filter for Microsoft Office and Corel Draw. Another application might use different fonts and/or different font names, which may not be documented. As a workaround, the ‘nofontlist‘ option deletes the font table from the CGM file. In this case, the reading application should use a default table. Gnuplot will still use its own default font table to select font indices. Thus, 'Helvetica' will give you an index of 1, which should get you the first entry in your application's default font table. 'Helvetica Bold' will give you its second entry, etc.

The former ‘winword6‘ option is now a deprecated synonym for ‘nofontlist‘. The problems involving the color and font tables that the ‘winword6‘ option was intended to work around turned out to be gnuplot bugs which have now been fixed."

3.1.0.11 corel

The ‘corel‘ terminal driver supports CorelDraw.

Syntax:

```
set terminal corel {  default
                   | {monochrome | color
                        {"<font>" {<fontsize>
                           {<xsize> <ysize> {<linewidth> }}}}}
```

where the fontsize and linewidth are specified in points and the sizes in inches. The defaults are monochrome, "SwitzerlandLight", 22, 8.2, 10 and 1.2."

3.1.0.12 debug

This terminal is provided to allow for the debugging of ‘gnuplot‘. It is likely to be of use only for users who are modifying the source code."

3.1.0.13 svga

The ‘svga‘ terminal driver supports PCs with SVGA graphics. It can only be used if it is compiled with DJGPP. Its only option is the font.

Syntax:

```
set terminal svga {"<fontname>"}"
```

3.1.0.14 dumb

The ‘dumb‘ terminal driver has an optional size specification and trailing linefeed control.

Syntax:

```
set terminal dumb {[no]feed} {<xsize> <ysize>}
                  {[no]enhanced}
```

where <xsize> and <ysize> set the size of the dumb terminals. Default is 79 by 24. The last newline is printed only if ‘feed‘ is enabled.

Examples:

```
set term dumb nofeed
set term dumb 79 49 # VGA screen---why would anyone do that?"
```

3.1.0.15 dxf

The ‘dxf‘ terminal driver creates pictures that can be imported into AutoCad (Release 10.x). It has no options of its own, but some features of its plots may be modified by other means. The default size is 120x80 AutoCad units, which can be changed by Section 2.21.57 [size], page 129. ‘dxf‘ uses seven colors (white, red, yellow, green, cyan, blue and magenta), which can be changed only by modifying the source file. If a black-and-white plotting device is used, the colors are mapped to differing line thicknesses. See the description of the AutoCad print/plot command."

3.1.0.16 dxy800a

This terminal driver supports the Roland DXY800A plotter. It has no options."

3.1.0.17 eepic

The ‘eepic‘ terminal driver supports the extended LaTeX picture environment. It is an alternative to the ‘latex‘ driver.

The output of this terminal is intended for use with the "eepic.sty" macro package for LaTeX. To use it, you need "eepic.sty", "epic.sty" and a printer driver that supports the "tpic" \\specials. If your printer driver doesn't support those \\specials, "eepicemu.sty" will enable you to use some of them. dvips and dvipdfm do support the "tpic" \\specials.

Syntax:

```
set terminal eepic {color, dashed, rotate, small, tiny, de-
fault, <fontsize>}
```

Options: You can give options in any order you wish. 'color' causes gnuplot to produce \\color{...} commands so that the graphs are colored. Using this option, you must include \\usepackage{color} in the preambel of your latex document. 'dashed' will allow dashed line types; without this option, only solid lines with varying thickness will be used. 'dashed' and 'color' are mutually exclusive; if 'color' is specified, then 'dashed' will be ignored. 'rotate' will enable true rotated text (by 90 degrees). Otherwise, rotated text will be typeset with letters stacked above each other. If you use this option you must include \\usepackage{graphicx} in the preamble. 'small' will use \\scriptsize symbols as point markers (Probably does not work with TeX, only LaTeX2e). Default is to use the default math size. 'tiny' uses \\scriptscriptstyle symbols. 'default' resets all options to their defaults = no color, no dashed lines, pseudo-rotated (stacked) text, large point symbols. <fontsize> is a number which specifies the font size inside the picture environment; the unit is pt (points), i.e., 10 pt equals approx. 3.5 mm. If fontsize is not specified, then all text inside the picture will be set in \\footnotesize.

Notes: Remember to escape the # character (or other chars meaningful to (La-)TeX) by \\\\ (2 backslashes). It seems that dashed lines become solid lines when the vertices of a plot are too close. (I do not know if that is a general problem with the tpic specials, or if it is caused by a bug in eepic.sty or dvips/dvipdfm.) The default size of an eepic plot is 5x3 inches, which can be scaled by 'set size a,b'. Points, among other things, are drawn using the LaTeX commands "\\Diamond", "\\Box", etc. These commands no longer belong to the LaTeX2e core; they are included in the latexsym package, which is part of the base distribution and thus part of any LaTeX implementation. Please do not forget to use this package. Instead of latexsym, you can also include the amssymb package. All drivers for LaTeX offer a special way of controlling text positioning: If any text string begins with '{', you also need to include a '}' at the end of the text, and the whole text will be centered both horizontally and vertically. If the text string begins with '[', you need to follow this with a position specification (up to two out of t,b,l,r), ']{', the text itself, and finally '}'. The text itself may be anything LaTeX can typeset as an LR-box. '\\rule{}{}'s may help for best positioning.

Examples: set term eepic

```
output graphs as eepic macros inside a picture environment;
\\input the resulting file in your LaTeX document.
```

set term eepic color tiny rotate 8

```
eepic macros with \\color macros, \\scripscriptsize point markers,
true rotated text, and all text set with 8pt.
```

About label positioning: Use gnuplot defaults (mostly sensible, but sometimes not really best):

```
set title '\\LaTeX\\ -- $ \\gamma $'
```

Force centering both horizontally and vertically:

```
set label '{\\LaTeX\\ -- $ \\gamma $}' at 0,0
```

Specify own positioning (top here):

```
set xlabel '[t]{\\LaTeX\\ -- $ \\gamma $}'
```

The other label – account for long ticlabels:

```
set ylabel '[r]{\\LaTeX\\ -- $ \\gamma $\\rule{7mm}{0pt}}'"
```

3.1.0.18 emf

The ‘emf‘ terminal generates an Enhanced Metafile Format file. This file format is the metafile standard on MS Win32 Systems.

Syntax:

```
set terminal emf {<color>} {solid | dashed} {linewidth <X>}
                 {"<font>"} {<fontsize>}
```

<color> is either ‘color‘ or ‘monochrome‘; ‘solid‘ draws all curves with solid lines, overriding any dashed patterns; ‘linewidth <factor>‘ multiplies all line widths by this factor. <font> is the name of a font; and ‘<fontsize>‘ is the size of the font in points.

The first two options can be in any order. Selecting ‘default‘ sets all options to their default values.

Examples:

```
set terminal emf 'Times Roman Italic' 12
set terminal emf color solid    # no pesky dashes!"
```

3.1.0.19 emxvga

The ‘emxvga‘, ‘emxvesa‘ and ‘vgal‘ terminal drivers support PCs with SVGA, vesa SVGA and VGA graphics boards, respectively. They are intended to be compiled with "emx-gcc" under either DOS or OS/2. They also need VESA and SVGAKIT maintained by Johannes Martin (JMARTIN@GOOFY.ZDV.UNI-MAINZ.DE) with additions by David J. Liu (liu@phri.nyu.edu).

Syntax:

```
set terminal emxvga
set terminal emxvesa {vesa-mode}
set terminal vgal
```

The only option is the vesa mode for ‘emxvesa‘, which defaults to G640x480x256."

3.1.0.20 epson-180dpi

This driver supports a family of Epson printers and derivatives.

‘epson-180dpi‘ and ‘epson-60dpi‘ are drivers for Epson LQ-style 24-pin printers with resolutions of 180 and 60 dots per inch, respectively.

'epson-lx800' is a generic 9-pin driver appropriate for printers like the Epson LX-800, the Star NL-10 and NX-1000, the PROPRINTER, and so forth.

'nec-cp6' is generic 24-pin driver that can be used for printers like the NEC CP6 and the Epson LQ-800.

The 'okidata' driver supports the 9-pin OKIDATA 320/321 Standard printers.

The 'starc' driver is for the Star Color Printer.

The 'tandy-60dpi' driver is for the Tandy DMP-130 series of 9-pin, 60-dpi printers.

Only 'nec-cp6' has any options.

Syntax:

```
set terminal nec-cp6 {monochrome | colour | draft}
```

which defaults to monochrome.

With each of these drivers, a binary copy is required on a PC to print. Do not use 'print'—use instead 'copy file /b lpt1:'."

3.1.0.21 excl

The 'excl' terminal driver supports Talaris printers such as the EXCL Laser printer and the 1590. It has no options."

3.1.0.22 hercules

These drivers supports PC monitors with autodetected graphics boards. They can be used only when compiled with Zortech C/C++. None have options."

3.1.0.23 fig

The 'fig' terminal device generates output in the Fig graphics language.

Syntax:

```
set terminal fig {monochrome | color}
                 {landscape | portrait}
                 {small | big | size <xsize> <ysize>}
                 {metric | inches}
                 {pointsmax <max_points>}
                 {solid | dashed}
                 {fontsize <fsize>}
                 {textnormal | {textspecial texthid-
   den textrigid}}
                 {{thickness|linewidth} <units>}
                 {depth <layer>}
                 {version <number>}
```

'monochrome' and 'color' determine whether the picture is black-and-white or 'color'. 'small' and 'big' produce a 5x3 or 8x5 inch graph in the default 'landscape' mode and 3x5 or 5x8 inches in 'portrait' mode. Section 2.21.57 [size], page 129 sets (overrides) the size of the drawing area to <xsize>*<ysize> in units of inches or centimeters depending on the ‘inches‘ or ‘metric‘ setting in effect. The latter settings is also used as default units for editing with "xfig".

‘pointsmax <max_points>‘ sets the maximum number of points per polyline.

‘solid‘ inhibits automatic usage of ‘dash‘ed lines when solid linestyles are used up, which otherwise occurs.

‘fontsize‘ sets the size of the text font to <fsize> points. ‘textnormal‘ resets the text flags and selects postscript fonts, ‘textspecial‘ sets the text flags for LaTeX specials, ‘texthidden‘ sets the hidden flag and ‘textrigid‘ the rigid flag.

‘depth‘ sets the default depth layer for all lines and text. The default depth is 10 to leave room for adding material with "xfig" on top of the plot.

Section 2.21.73 [version], page 150 sets the format version of the generated fig output. Currently only versions 3.1 and 3.2 are supported.

‘thickness‘ sets the default line thickness, which is 1 if not specified. Overriding the thickness can be achieved by adding a multiple of 100 to the ‘linetype‘ value for a ‘plot‘ command. In a similar way the ‘depth‘ of plot elements (with respect to the default depth) can be controlled by adding a multiple of 1000 to <linetype>. The depth is then <layer> + <linetype>/1000 and the thickness is (<linetype>%1000)/100 or, if that is zero, the default line thickness. ‘linewidth‘ is a synonym for ‘thickness‘.

Additional point-plot symbols are also available with the ‘fig‘ driver. The symbols can be used through ‘pointtype‘ values % 100 above 50, with different fill intensities controlled by <pointtype> % 5 and outlines in black (for <pointtype> % 10 < 5) or in the current color. Available symbols are

```
50 - 59:  circles
60 - 69:  squares
70 - 79:  diamonds
80 - 89:  upwards triangles
90 - 99:  downwards triangles
```

The size of these symbols is linked to the font size. The depth of symbols is by default one less than the depth for lines to achieve nice error bars. If <pointtype> is above 1000, the depth is <layer> + <pointtype>/1000-1. If <pointtype>%1000 is above 100, the fill color is (<pointtype>%1000)/100-1.

Available fill colors are (from 1 to 9): black, blue, green, cyan, red, magenta, yellow, white and dark blue (in monochrome mode: black for 1 to 6 and white for 7 to 9).

See Section 2.12.7 [with], page 68 for details of <linetype> and <pointtype>.

The ‘big‘ option is a substitute for the ‘bfig‘ terminal in earlier versions, which is no longer supported.

Examples:

```
set terminal fig monochrome small pointsmax 1000  # defaults

plot 'file.dat' with points linetype 102 pointtype 759
```

would produce circles with a blue outline of width 1 and yellow fill color.

```
plot 'file.dat' using 1:2:3 with err linetype 1 pointtype 554
```

would produce errorbars with black lines and circles filled red. These circles are one layer above the lines (at depth 9 by default).

To plot the error bars on top of the circles use

```
plot 'file.dat' using 1:2:3 with err linetype 1 pointtype 2554"
```

3.1.0.24 png (NEW)

Syntax:

```
set terminal png
       {{no}transparent} {{no}interlace}
       {{no}truecolor} {rounded|butt}
       {tiny | small | medium | large | giant}
       {font <face> {<pointsize>}}
       {size <x>,<y>} {{no}crop}
       {{no}enhanced}
       {<color0> <color1> <color2> ...}
```

PNG images are created using libgd, with optional support for TrueType and Adobe Type 1 fonts via libfreetype. Version 1.8 or greater of libgd is required.

'transparent' instructs the driver to generate transparent PNGs. The first color will be the transparent one. Default is 'notransparent'.

'interlace' instructs the driver to generate interlaced PNGs. Default is 'nointerlace'.

'butt' instructs the driver to use a line drawing method that does not overshoot the desired end point of a line. This setting is only applicable for line widths greater than 1. This setting is most useful when drawing horizontal or vertical lines. Default is 'rounded'. Version 2.0 or greater of libgd is required.

PNG plots may be conveniently viewed by piping the output to the 'display' program from the ImageMagick package as follows:

```
set term png
set output '| display png:-'
```

View the output from successive plot commands interactively by hitting <space> in the display window. To save a particular one to disk, left click in the display window and choose Section 2.20 [save], page 73.

Five basic fonts are supported directly by the gd library. These are 'tiny' (5x8 pixels), 'small' (6x12 pixels), 'medium', (7x13 Bold), 'large' (8x16) or 'giant' (9x15 pixels). These fonts cannot be scaled or rotated (pure horizontal or vertical text only).

If gnuplot was built with support for TrueType (*.ttf) or Adobe Type 1 (*.pfa) fonts, they may be selected using the 'font <face> {<pointsize>}' option. <face> is either the full pathname to the font file, or a font face name that is assumed to be the first part of a filename in one of the directories listed in the GDFONTPATH environmental variable. That is, 'set term png font "Face"' will look for a font file named either <somedirectory>/Face.ttf or <somedirectory>/Face.pfa. Both TrueType and Adobe Type 1 fonts are fully scalable and may be rotated through any angle. If no font is specified, gnuplot checks the environmental variable GNUPLOT_DEFAULT_GDFONT to see if there is a preferred default font.

‘enhanced‘ enables the enhanced text processing features, (subscripts, superscripts and mixed fonts). See ‘enhanced‘ for more information. The full enhanced mode syntax is supported by the PNG/JPEG driver itself, but some of these features are dependent on which version of the underlying libgd library is present, and which fonts are available.

The size <x,y> is given in pixels—it defaults to 640x480. The number of pixels can be also modified by scaling with the Section 2.21.57 [size], page 129 command. ‘crop‘ trims blank space from the edges of the completed plot, resulting in a smaller final image size. Default is ‘nocrop‘.

Each color must be of the form 'xrrggbb', where x is the literal character 'x' and 'rrggbb' are the red, green and blue components in hex. For example, 'x00ff00' is green. The background color is set first, then the border colors, then the X & Y axis colors, then the plotting colors. The maximum number of colors that can be set is 256.

Examples:

```
      set terminal png medium size 640,480 \\
                       xffffff x000000 x404040 \\
                       xff0000 xffa500 x66cdaa xcdb5cd \\
                       xadd8e6 x0000ff xdda0dd x9500d3    # defaults
```

which uses white for the non-transparent background, black for borders, gray for the axes, and red, orange, medium aquamarine, thistle 3, light blue, blue, plum and dark violet for eight plotting colors.

```
      set terminal png font arial 14 size 800,600
```

which searches for a TrueType font with face name 'arial' in the directory specified by the environment variable GDFONTPATH and 14pt font size.

```
      set terminal png transparent xffffff \\
                       x000000 x202020 x404040 x606060 \\
                       x808080 xA0A0A0 xC0C0C0 xE0E0E0
```

which uses white for the transparent background, black for borders, dark gray for axes, and a gray-scale for the six plotting colors.

3.1.0.25 ggi

The ‘ggi‘ driver can run on different targets as X or svgalib.

Syntax:

```
   set terminal ggi [acceleration <integer>] [[mode] {mode}]
```

In X the window cannot be resized using window manager handles, but the mode can be given with the mode option, e.g.:

```
 - V1024x768
 - V800x600
 - V640x480
```

```
- V320x200
```

Please refer to the ggi documentation for other modes. The 'mode' keyword is optional. It is recommended to select the target by environment variables as explained in the libggi manual page. To get DGA on X, you should for example

```
bash> export GGI_DISPLAY=DGA
csh>  setenv GGI_DISPLAY DGA
```

'acceleration' is only used for targets which report relative pointer motion events (e.g. DGA) and is a strictly positive integer multiplication factor for the relative distances. The default for acceleration is 7.

Examples:

```
set term ggi acc 10
set term ggi acc 1 mode V1024x768
set term ggi V1024x768"
```

3.1.0.26 Gnugraph(GNU plotutils)

The 'gnugraph' driver produces device-independent output in the GNU plot graphics language. The default size of the PostScript results generated by "plot2ps" is 5 x 3 inches; this can be increased up to about 8.25 x 8.25 by Section 2.21.57 [size], page 129.

Syntax:

```
set terminal gnugraph {"<fontname>"} {<fontsize>}
                      {type <pt>} {size "<size>"}
```

which defaults to 10-point "Courier".

For 'type', the following options are accepted: 'X', 'pnm', 'gif', 'ai', 'ps', 'cgm', 'fig', 'pcl5', 'hpgl', 'tek', and 'meta' (default). The Section 2.21.57 [size], page 129 option (default is a4) is passed straight through to plotutils, it's the user's responsibility to provide correct values. Details can be found in the plotutils documentation.

Examples:

```
      set terminal gnugraph type hpgl size "a4"
      set terminal gnugraph size "a4,xoffset=-
5mm,yoffset=2.0cm" type pnm
```

There is a non-GNU version of the 'gnugraph' driver which cannot be compiled unless this version is left out."

3.1.0.27 gpic

The 'gpic' terminal driver generates GPIC graphs in the Free Software Foundations's "groff" package. The default size is 5 x 3 inches. The only option is the origin, which defaults to (0,0).

Syntax:

```
set terminal gpic {<x> <y>}
```

where ‘x‘ and ‘y‘ are in inches.

A simple graph can be formatted using

```
groff -p -mpic -Tps file.pic > file.ps.
```

The output from pic can be pipe-lined into eqn, so it is possible to put complex functions in a graph with the Section 2.21.29 [label], page 104 and ‘set {x/y}label‘ commands. For instance,

```
set ylab '@space 0 int from 0 to x alpha ( t ) roman d t@'
```

will label the y axis with a nice integral if formatted with the command:

```
gpic filename.pic | geqn -d@@ -Tps | groff -m[macro-package] -
Tps
    > filename.ps
```

Figures made this way can be scaled to fit into a document. The pic language is easy to understand, so the graphs can be edited by hand if need be. All co-ordinates in the pic-file produced by ‘gnuplot‘ are given as x+gnuplotx and y+gnuploty. By default x and y are given the value 0. If this line is removed with an editor in a number of files, one can put several graphs in one figure like this (default size is 5.0x3.0 inches):

```
.PS 8.0
x=0;y=3
copy "figa.pic"
x=5;y=3
copy "figb.pic"
x=0;y=0
copy "figc.pic"
x=5;y=0
copy "figd.pic"
.PE
```

This will produce an 8-inch-wide figure with four graphs in two rows on top of each other.

One can also achieve the same thing by the command

```
set terminal gpic x y
```

for example, using

```
.PS 6.0
copy "trig.pic"
.PE"
```

3.1.0.28 gpic

The 'gpic' terminal driver generates GPIC graphs in the Free Software Foundations's "groff" package. The default size is 5 x 3 inches. The only option is the origin, which defaults to (0,0).

Syntax:

```
set terminal gpic {<x> <y>}
```

where 'x' and 'y' are in inches.

A simple graph can be formatted using

```
groff -p -mpic -Tps file.pic > file.ps.
```

The output from pic can be pipe-lined into eqn, so it is possible to put complex functions in a graph with the Section 2.21.29 [label], page 104 and 'set {x/y}label' commands. For instance,

```
set ylab '@space 0 int from 0 to x alpha ( t ) roman d t@'
```

will label the y axis with a nice integral if formatted with the command:

```
gpic filename.pic | geqn -d@@ -Tps | groff -m[macro-package] -
Tps
        > filename.ps
```

Figures made this way can be scaled to fit into a document. The pic language is easy to understand, so the graphs can be edited by hand if need be. All co-ordinates in the pic-file produced by 'gnuplot' are given as x+gnuplotx and y+gnuploty. By default x and y are given the value 0. If this line is removed with an editor in a number of files, one can put several graphs in one figure like this (default size is 5.0x3.0 inches):

```
.PS 8.0
x=0;y=3
copy "figa.pic"
x=5;y=3
copy "figb.pic"
x=0;y=0
copy "figc.pic"
x=5;y=0
copy "figd.pic"
.PE
```

This will produce an 8-inch-wide figure with four graphs in two rows on top of each other.

One can also achieve the same thing by the command

```
set terminal gpic x y
```

for example, using

```
.PS 6.0
copy "trig.pic"
.PE"
```

3.1.0.29 gpr

The ‘gpr‘ terminal driver supports the Apollo Graphics Primitive Resource for a fixed-size window. It has no options.

If a variable window size is desired, use the ‘apollo‘ terminal instead."

3.1.0.30 grass

The ‘grass‘ terminal driver gives ‘gnuplot‘ capabilities to users of the GRASS geographic information system. Contact grassp-list@moon.cecer.army.mil for more information. Pages are written to the current frame of the GRASS Graphics Window. There are no options."

3.1.0.31 hp2623a

The ‘hp2623a‘ terminal driver supports the Hewlett Packard HP2623A. It has no options."

3.1.0.32 hp2648

The ‘hp2648‘ terminal driver supports the Hewlett Packard HP2647 and HP2648. It has no options."

3.1.0.33 hp500c

The ‘hp500c‘ terminal driver supports the Hewlett Packard HP DeskJet 500c. It has options for resolution and compression.

Syntax:

```
set terminal hp500c {<res>} {<comp>}
```

where ‘res‘ can be 75, 100, 150 or 300 dots per inch and ‘comp‘ can be "rle", or "tiff". Any other inputs are replaced by the defaults, which are 75 dpi and no compression. Rasterization at the higher resolutions may require a large amount of memory."

3.1.0.34 hpgl

The ‘hpgl‘ driver produces HPGL output for devices like the HP7475A plotter. There are two options which can be set: the number of pens and ‘eject‘, which tells the plotter to eject a page when done. The default is to use 6 pens and not to eject the page when done.

The international character sets ISO-8859-1 and CP850 are recognized via ‘set encoding iso_8859_1‘ or ‘set encoding cp850‘ (see Section 2.21.18 [encoding], page 91 for details).

Syntax:

```
set terminal hpgl {<number_of_pens>} {eject}
```

The selection

```
set terminal hpgl 8 eject
```

is equivalent to the previous ‘hp7550‘ terminal, and the selection

```
set terminal hpgl 4
```

is equivalent to the previous 'hp7580b' terminal.

The 'pcl5' driver supports plotters such as the Hewlett-Packard Designjet 750C, the Hewlett-Packard Laserjet III, and the Hewlett-Packard Laserjet IV. It actually uses HPGL-2, but there is a name conflict among the terminal devices. It has several options which must be specified in the order indicated below:

Syntax:

```
set terminal pcl5 {mode <mode>} {<plotsize>}
    {{color {<number_of_pens>}} | monochrome} {solid | dashed}
    {font <font>} {size <fontsize>} {pspoints | nopspoints}
```

<mode> is 'landscape' or 'portrait'. <plotsize> is the physical plotting size of the plot, which is one of the following: 'letter' for standard (8 1/2" X 11") displays, 'legal' for (8 1/2" X 14") displays, 'noextended' for (36" X 48") displays (a letter size ratio) or, 'extended' for (36" X 55") displays (almost a legal size ratio). 'color' is for multi-pen (i.e. color) plots, and <number_of_pens> is the number of pens (i.e. colors) used in color plots. 'monochrome' is for one (e.g. black) pen plots. 'solid' draws all lines as solid lines, or 'dashed' will draw lines with different dashed and dotted line patterns. <font> is 'stick', 'univers', 'cg_times', 'zapf_dingbats', 'antique_olive', 'arial', 'courier', 'garamond_antigua', 'letter_gothic', 'cg_omega', 'albertus', 'times_new_roman', 'clarendon', 'coronet', 'marigold', 'truetype_symbols', or 'wingdings'. <fontsize> is the font size in points. The point type selection can be the standard default set by specifying 'nopspoints', or the same set of point types found in the postscript terminal by specifying 'pspoints'.

Note that built-in support of some of these options is printer device dependent. For instance, all the fonts are supposedly supported by the HP Laserjet IV, but only a few (e.g. univers, stick) may be supported by the HP Laserjet III and the Designjet 750C. Also, color obviously won't work on the the laserjets since they are monochrome devices.

Defaults: landscape, noextended, color (6 pens), solid, univers, 12 point,

```
    and nopspoints.
```

With 'pcl5' international characters are handled by the printer; you just put the appropriate 8-bit character codes into the text strings. You don't need to bother with Section 2.21.18 [encoding], page 91.

HPGL graphics can be imported by many software packages."

3.1.0.35 hpljii

The 'hpljii' terminal driver supports the HP Laserjet Series II printer. The 'hpdj' driver supports the HP DeskJet 500 printer. These drivers allow a choice of resolutions.

Syntax:

```
set terminal hpljii | hpdj {<res>}
```

where 'res' may be 75, 100, 150 or 300 dots per inch; the default is 75. Rasterization at the higher resolutions may require a large amount of memory.

The ‘hp500c‘ terminal is similar to ‘hpdj‘; ‘hp500c‘ additionally supports color and compression."

3.1.0.36 hppj

The ‘hppj‘ terminal driver supports the HP PaintJet and HP3630 printers. The only option is the choice of font.

Syntax:

```
set terminal hppj {FNT5X9 | FNT9X17 | FNT13X25}
```

with the middle-sized font (FNT9X17) being the default."

3.1.0.37 imagen

The ‘imagen‘ terminal driver supports Imagen laser printers. It is capable of placing multiple graphs on a single page.

Syntax:

```
set terminal imagen {<fontsize>} {portrait | landscape}
                    {[<horiz>,<vert>]}
```

where ‘fontsize‘ defaults to 12 points and the layout defaults to ‘landscape‘. ‘<horiz>‘ and ‘<vert>‘ are the number of graphs in the horizontal and vertical directions; these default to unity.

Example:

```
set terminal imagen portrait [2,3]
```

puts six graphs on the page in three rows of two in portrait orientation."

3.1.0.38 iris4d

The ‘iris4d‘ terminal driver supports Silicon Graphics IRIS 4D computers. Its only option is 8- or 24-bit color depth. The default is 8.

Syntax:

```
set terminal iris4d {8 | 24}
```

The color depth is not really a choice – the value appropriate for the hardware should be selected.

When using 24-bit mode, the colors can be directly specified via the file .gnuplot_iris4d that is searched in the current directory and then in the home directory specified by the HOME environment variable. This file holds RGB values for the background, border, labels and nine plotting colors, in that order. For example, here is a file containing the default colors:

```
85   85   85     Background   (dark gray)
0    0    0      Boundary     (black)
170  0    170    Labeling     (magenta)
85   255  255    Plot Color 1 (light cyan)
```

```
170  0    0      Plot Color 2 (red)
0    170  0      Plot Color 3 (green)
255  85   255    Plot Color 4 (light magenta)
255  255  85     Plot Color 5 (yellow)
255  85   85     Plot Color 6 (light red)
85   255  85     Plot Color 7 (light green)
0    170  170    Plot Color 8 (cyan)
170  170  0      Plot Color 9 (brown)
```

This file must have exactly 12 lines of RGB triples. No empty lines are allowed, and anything after the third number on a line is ignored."

3.1.0.39 kyo

The ‘kyo‘ and ‘prescribe‘ terminal drivers support the Kyocera laser printer. The only difference between the two is that ‘kyo‘ uses "Helvetica" whereas ‘prescribe‘ uses "Courier". There are no options."

3.1.0.40 latex

The ‘latex‘ and ‘emtex‘ drivers allow two options.

Syntax:

```
set terminal latex | emtex {courier | roman | de-
fault} {<fontsize>}
```

‘fontsize‘ may be any size you specify. The default is for the plot to inherit its font setting from the embedding document.

Unless your driver is capable of building fonts at any size (e.g. dvips), stick to the standard 10, 11 and 12 point sizes.

METAFONT users beware: METAFONT does not like odd sizes.

All drivers for LaTeX offer a special way of controlling text positioning: If any text string begins with '{', you also need to include a '}' at the end of the text, and the whole text will be centered both horizontally and vertically. If the text string begins with '[', you need to follow this with a position specification (up to two out of t,b,l,r), ']{', the text itself, and finally '}'. The text itself may be anything LaTeX can typeset as an LR-box. '\\rule{}{}'s may help for best positioning.

Points, among other things, are drawn using the LaTeX commands "\\Diamond" and "\\Box". These commands no longer belong to the LaTeX2e core; they are included in the latexsym package, which is part of the base distribution and thus part of any LaTeX implementation. Please do not forget to use this package.

Examples: About label positioning: Use gnuplot defaults (mostly sensible, but sometimes not really best):

```
set title '\\LaTeX\\ -- $ \\gamma $'
```

Force centering both horizontally and vertically:

```
set label '{\\LaTeX\\ -- $ \\gamma $}' at 0,0
```

Specify own positioning (top here):

```
set xlabel '[t]{\\LaTeX\\ -- $ \\gamma $}'
```

The other label – account for long ticlabels:

```
set ylabel '[r]{\\LaTeX\\ -- $ \\gamma $\\rule{7mm}{0pt}}'"
```

3.1.0.41 linux

The 'linux' driver has no additional options to specify. It looks at the environment variable GSVGAMODE for the default mode; if not set, it uses 1024x768x256 as default mode or, if that is not possible, 640x480x16 (standard VGA)."

3.1.0.42 linux

The 'linux' driver has no additional options to specify. It looks at the environment variable GSVGAMODE for the default mode; if not set, it uses 1024x768x256 as default mode or, if that is not possible, 640x480x16 (standard VGA)."

3.1.0.43 macintosh

Several options may be set in the 'macintosh' driver.

Syntax:

```
     set terminal macintosh {singlewin | multiwin} {verti-
cal | novertical}
                               {size <width>, <height> | default}
```

'singlewin' limits the output to a single window and is useful for animations. 'multiwin' allows multiple windows. 'vertical' is only valid under the gx option. With this option, rotated text

```
    be drawn vertically. novertical turns this option off.
 size <width>, <height> over-
rides the graph size set in the preferences
    dialog until it is cleared with either 'set term mac size default'
    or 'set term mac default'.

 'set term mac size default' sets the window size set-
tings to those set in
    the preferences dialog.

 'set term mac default' sets all options to their default values.
    Default values: nogx, multiwin, novertical.

 If you generate graphs under the multiwin op-
tion and then switch to singlewin,
 the next plot command will cause one more window to be cre-
ated. This new
 window will be reused as long as singlewin is in ef-
fect. If you switch back
```

```
 to multiwin, generate some graphs, and then switch to sin-
glewin again, the
 orginal 'singlewin' window will be re-
sused if it is still open. Otherwise
 a new 'singlewin' window will be created. The 'singlewin' win-
dow is not numbered."
```

3.1.0.44 mf

The 'mf' terminal driver creates an input file to the METAFONT program. Thus a figure may be used in the TeX document in the same way as is a character.

To use a picture in a document, the METAFONT program must be run with the output file from 'gnuplot' as input. Thus, the user needs a basic knowledge of the font creating process and the procedure for including a new font in a document. However, if the METAFONT program is set up properly at the local site, an unexperienced user could perform the operation without much trouble.

The text support is based on a METAFONT character set. Currently the Computer Modern Roman font set is input, but the user is in principal free to choose whatever fonts he or she needs. The METAFONT source files for the chosen font must be available. Each character is stored in a separate picture variable in METAFONT. These variables may be manipulated (rotated, scaled etc.) when characters are needed. The drawback is the interpretation time in the METAFONT program. On some machines (i.e. PC) the limited amount of memory available may also cause problems if too many pictures are stored.

The 'mf' terminal has no options.

— METAFONT INSTRUCTIONS —

- Set your terminal to METAFONT:

```
set terminal mf
```

- Select an output-file, e.g.:

```
set output "myfigures.mf"
```

- Create your pictures. Each picture will generate a separate character. Its default size will be 5*3 inches. You can change the size by saying 'set size 0.5,0.5' or whatever fraction of the default size you want to have.

- Quit 'gnuplot'.

- Generate a TFM and GF file by running METAFONT on the output of 'gnuplot'. Since the picture is quite large (5*3 in), you will have to use a version of METAFONT that has a value of at least 150000 for memmax. On Unix systems these are conventionally installed under the name bigmf. For the following assume that the command virmf stands for a big version of METAFONT. For example:

- Invoke METAFONT:

```
virmf '&plain'
```

- Select the output device: At the METAFONT prompt ('*') type:

```
\\mode:=CanonCX;     % or whatever printer you use
```

- Optionally select a magnification:

```
mag:=1;              % or whatever you wish
```

- Input the ‘gnuplot‘-file:

```
input myfigures.mf
```

On a typical Unix machine there will usually be a script called "mf" that executes virmf '&plain', so you probably can substitute mf for virmf &plain. This will generate two files: mfput.tfm and mfput.$$$gf (where $$$ indicates the resolution of your device). The above can be conveniently achieved by typing everything on the command line, e.g.: virmf '&plain' '\\mode:=CanonCX; mag:=1; input myfigures.mf' In this case the output files will be named myfigures.tfm and myfigures.300gf.

- Generate a PK file from the GF file using gftopk:

```
gftopk myfigures.300gf myfigures.300pk
```

The name of the output file for gftopk depends on the DVI driver you use. Ask your local TeX administrator about the naming conventions. Next, either install the TFM and PK files in the appropriate directories, or set your environment variables properly. Usually this involves setting TEXFONTS to include the current directory and doing the same thing for the environment variable that your DVI driver uses (no standard name here...). This step is necessary so that TeX will find the font metric file and your DVI driver will find the PK file.

- To include your pictures in your document you have to tell TeX the font:

```
\\font\\gnufigs=myfigures
```

Each picture you made is stored in a single character. The first picture is character 0, the second is character 1, and so on... After doing the above step, you can use the pictures just like any other characters. Therefore, to place pictures 1 and 2 centered in your document, all you have to do is:

```
\\centerline{\\gnufigs\\char0}
\\centerline{\\gnufigs\\char1}
```

in plain TeX. For LaTeX you can, of course, use the picture environment and place the picture wherever you wish by using the \\makebox and \\put macros.

This conversion saves you a lot of time once you have generated the font; TeX handles the pictures as characters and uses minimal time to place them, and the documents you make change more often than the pictures do. It also saves a lot of TeX memory. One last advantage of using the METAFONT driver is that the DVI file really remains device independent, because no \\special commands are used as in the eepic and tpic drivers."

3.1.0.45 mp

The ‘mp‘ driver produces output intended to be input to the Metapost program. Running Metapost on the file creates EPS files containing the plots. By default, Metapost passes all text through TeX. This has the advantage of allowing essentially any TeX symbols in titles and labels.

Syntax:

```
set term mp {color | colour | monochrome}
            {solid | dashed}
            {notex | tex | latex}
            {magnification <magsize>}
```

```
{psnfss | psnfss-version7 | nopsnfss}
{prologues <value>}
{a4paper}
{amstex}
{"<fontname>"} {<fontsize>}
```

The option ‘color‘ causes lines to be drawn in color (on a printer or display that supports it), ‘monochrome‘ (or nothing) selects black lines. The option ‘solid‘ draws solid lines, while ‘dashed‘ (or nothing) selects lines with different patterns of dashes. If ‘solid‘ is selected but ‘color‘ is not, nearly all lines will be identical. This may occasionally be useful, so it is allowed.

The option ‘notex‘ bypasses TeX entirely, therefore no TeX code can be used in labels under this option. This is intended for use on old plot files or files that make frequent use of common characters like ‘$‘ and ‘%‘ that require special handling in TeX.

The option ‘tex‘ sets the terminal to output its text for TeX to process.

The option ‘latex‘ sets the terminal to output its text for processing by LaTeX. This allows things like \\frac for fractions which LaTeX knows about but TeX does not. Note that you must set the environment variable TEX to the name of your LaTeX executable (normally latex) if you use this option or use ‘mpost –tex=<name of LaTeX executable> ...‘. Otherwise metapost will try and use TeX to process the text and it won’t work.

Changing font sizes in TeX has no effect on the size of mathematics, and there is no foolproof way to make such a change, except by globally setting a magnification factor. This is the purpose of the ‘magnification‘ option. It must be followed by a scaling factor. All text (NOT the graphs) will be scaled by this factor. Use this if you have math that you want at some size other than the default 10pt. Unfortunately, all math will be the same size, but see the discussion below on editing the MP output. ‘mag‘ will also work under ‘notex‘ but there seems no point in using it as the font size option (below) works as well.

The option ‘psnfss‘ uses postscript fonts in combination with LaTeX. Since this option only makes sense, if LaTeX is being used, the ‘latex‘ option is selected automatically. This option includes the following packages for LaTeX: inputenc(latin1), fontenc(T1), mathptmx, helvet(scaled=09.2), courier, latexsym and textcomp.

The option ‘psnfss-version7‘ uses also postscript fonts in LaTeX (option ‘latex‘ is also automatically selected), but uses the following packages with LaTeX: inputenc(latin1), fontenc(T1), times, mathptmx, helvet and courier.

The option ‘nopsnfss‘ is the default and uses the standard font (cmr10 if not otherwise specified).

The option ‘prologues‘ takes a value as an additional argument and adds the line ‘prologues:=<value>‘ to the metapost file. If a value of ‘2‘ is specified metapost uses postscript fonts to generate the eps-file, so that the result can be viewed using e.g. ghostscript. Normally the output of metapost uses TeX fonts and therefore has to be included in a (La)TeX file before you can look at it.

The option ‘noprologues‘ is the default. No additional line specifying the prologue will be added.

The option ‘a4paper‘ adds a ‘[a4paper]‘ to the documentclass. Normally letter paper is used (default). Since this option is only used in case of LaTeX, the ‘latex‘ option is selected automatically.

The option ‘amstex‘ automatically selects the ‘latex‘ option and includes the following LaTeX packages: amsfonts, amsmath(intlimits). By default these packages are not included.

A name in quotes selects the font that will be used when no explicit font is given in a Section 2.21.29 [label], page 104 or ‘set title‘. A name recognized by TeX (a TFM file exists) must be used. The default is "cmr10" unless ‘notex‘ is selected, then it is "pcrr8r" (Courier). Even under ‘notex‘, a TFM file is needed by Metapost. The file ‘pcrr8r.tfm‘ is the name given to Courier in LaTeX’s psnfss package. If you change the font from the ‘notex‘ default, choose a font that matches the ASCII encoding at least in the range 32-126. ‘cmtt10‘ almost works, but it has a nonblank character in position 32 (space).

The size can be any number between 5.0 and 99.99. If it is omitted, 10.0 is used. It is advisable to use ‘magstep‘ sizes: 10 times an integer or half-integer power of 1.2, rounded to two decimals, because those are the most available sizes of fonts in TeX systems.

All the options are optional. If font information is given, it must be at the end, with size (if present) last. The size is needed to select a size for the font, even if the font name includes size information. For example, ‘set term mp "cmtt12"‘ selects cmtt12 shrunk to the default size 10. This is probably not what you want or you would have used cmtt10.

The following common ascii characters need special treatment in TeX:

```
$, &, #, %, _;  |, <, >;  ^, ~,  \\, {, and }
```

The five characters $, #, &, _, and % can simply be escaped, e.g., ‘\\$‘. The three characters <, >, and | can be wrapped in math mode, e.g., ‘$<$‘. The remainder require some TeX work-arounds. Any good book on TeX will give some guidance.

If you type your labels inside double quotes, backslashes in TeX code need to be escaped (doubled). Using single quotes will avoid having to do this, but then you cannot use ‘\\n‘ for line breaks. As of this writing, version 3.7 of gnuplot processes titles given in a ‘plot‘ command differently than in other places, and backslashes in TeX commands need to be doubled regardless of the style of quotes.

Metapost pictures are typically used in TeX documents. Metapost deals with fonts pretty much the same way TeX does, which is different from most other document preparation programs. If the picture is included in a LaTeX document using the graphics package, or in a plainTeX document via epsf.tex, and then converted to PostScript with dvips (or other dvi-to-ps converter), the text in the plot will usually be handled correctly. However, the text may not appear if you send the Metapost output as-is to a PostScript interpreter.

— METAPOST INSTRUCTIONS —

- Set your terminal to Metapost, e.g.:

```
set terminal mp mono "cmtt12" 12
```

- Select an output-file, e.g.:

```
set output "figure.mp"
```

- Create your pictures. Each plot (or multiplot group) will generate a separate Metapost beginfig...endfig group. Its default size will be 5 by 3 inches. You can change the size by saying ‘set size 0.5,0.5‘ or whatever fraction of the default size you want to have.

- Quit gnuplot.

- Generate EPS files by running Metapost on the output of gnuplot:

```
mpost figure.mp  OR  mp figure.mp
```

The name of the Metapost program depends on the system, typically ‘mpost‘ for a Unix machine and ‘mp‘ on many others. Metapost will generate one EPS file for each picture.

- To include your pictures in your document you can use the graphics package in LaTeX or epsf.tex in plainTeX:

```
\\usepackage{graphics} % LaTeX
\\input epsf.tex       % plainTeX
```

If you use a driver other than dvips for converting TeX DVI output to PS, you may need to add the following line in your LaTeX document:

```
\\DeclareGraphicsRule{*}{eps}{*}{}
```

Each picture you made is in a separate file. The first picture is in, e.g., figure.0, the second in figure.1, and so on.... To place the third picture in your document, for example, all you have to do is:

```
\\includegraphics{figure.2} % LaTeX
\\epsfbox{figure.2}         % plainTeX
```

The advantage, if any, of the mp terminal over a postscript terminal is editable output. Considerable effort went into making this output as clean as possible. For those knowledgeable in the Metapost language, the default line types and colors can be changed by editing the arrays ‘lt[]‘ and ‘col[]‘. The choice of solid vs dashed lines, and color vs black lines can be change by changing the values assigned to the booleans ‘dashedlines‘ and ‘colorlines‘. If the default ‘tex‘ option was in effect, global changes to the text of labels can be achieved by editing the ‘vebatimtex...etex‘ block. In particular, a LaTeX preamble can be added if desired, and then LaTeX’s built-in size changing commands can be used for maximum flexibility. Be sure to set the appropriate MP configuration variable to force Metapost to run LaTeX instead of plainTeX."

3.1.0.46 mgr

The ‘mgr‘ terminal driver supports the Mgr Window system. It has no options."

3.1.0.47 mif

The ‘mif‘ terminal driver produces Frame Maker MIF format version 3.00. It plots in MIF Frames with the size 15*10 cm, and plot primitives with the same pen will be grouped in the same MIF group. Plot primitives in a ‘gnuplot‘ page will be plotted in a MIF Frame, and several MIF Frames are collected in one large MIF Frame. The MIF font used for text is "Times".

Several options may be set in the MIF 3.00 driver.

Syntax:

```
        set terminal mif {color | colour | monochrome} {poly-
line | vectors}
                         {help | ?}
```

'colour' plots lines with line types >= 0 in colour (MIF sep. 2–7) and 'monochrome' plots all line types in black (MIF sep. 0). 'polyline' plots curves as continuous curves and 'vectors' plots curves as collections of vectors. Section 2.6 [help], page 45 and '¿ print online help on standard error output—both print a short description of the usage; Section 2.6 [help], page 45 also lists the options.

Examples:

```
        set term mif colour polylines    # defaults
        set term mif                     # defaults
        set term mif vectors
        set term mif help"
```

3.1.0.48 mtos

The 'mtos' terminal has no options. It sends data via a pipe to an external program called GPCLIENT. It runs under MULTITOS, Magic 3.x, MagicMAC. and MiNT. If you cannot find GPCLIENT, than mail to dirk@lstm.uni-erlangen.de."

3.1.0.49 next

Several options may be set in the next driver.

Syntax:

```
        set terminal next {<mode>} {<type> } {<color>} {<dashed>}
                 {"<fontname>"} {<fontsize>} title {"<newtitle>"}
```

where <mode> is 'default', which sets all options to their defaults; <type> is either 'new' or 'old', where 'old' invokes the old single window; <color> is either 'color' or 'monochrome'; <dashed> is either 'solid' or 'dashed'; "<fontname>" is the name of a valid PostScript font; <fontsize> is the size of the font in PostScript points; and <title> is the title for the GnuTerm window. Defaults are 'new', 'monochrome', 'dashed', "Helvetica", 14pt.

Examples:

```
        set term next default
        set term next 22
        set term next color "Times-Roman" 14
        set term next color "Helvetica" 12 title "MyPlot"
        set term next old
```

Pointsizes may be changed with 'set linestyle'."

3.1.0.50 Openstep (next)

/*

*/ Several options may be set in the openstep (next) driver.

Syntax:

```
set terminal openstep {<mode>} {<type> } {<color>} {<dashed>}
           {"<fontname>"} {<fontsize>} title {"<newtitle>"}
```

where <mode> is ‘default‘, which sets all options to their defaults; <type> is either ‘new‘ or ‘old‘, where ‘old‘ invokes the old single window; <color> is either ‘color‘ or ‘monochrome‘; <dashed> is either ‘solid‘ or ‘dashed‘; "<fontname>" is the name of a valid PostScript font; <fontsize> is the size of the font in PostScript points; and <title> is the title for the GnuTerm window. Defaults are ‘new‘, ‘monochrome‘, ‘dashed‘, "Helvetica", 14pt.

Examples:

```
set term openstep default
set term openstep 22
set term openstep color "Times-Roman" 14
set term openstep color "Helvetica" 12 title "MyPlot"
set term openstep old
```

Pointsizes may be changed with ‘set linestyle‘."

3.1.0.51 pbm

Several options may be set in the ‘pbm‘ terminal—the driver for PBMplus.

Syntax:

```
set terminal pbm {<fontsize>} {<mode>} {size <x>,<y>}
```

where <fontsize> is ‘small‘, ‘medium‘, or ‘large‘ and <mode> is ‘monochrome‘, ‘gray‘ or ‘color‘. The default plot size is 640 pixels wide and 480 pixels high.

The output of the ‘pbm‘ driver depends upon <mode>: ‘monochrome‘ produces a portable bitmap (one bit per pixel), ‘gray‘ a portable graymap (three bits per pixel) and ‘color‘ a portable pixmap (color, four bits per pixel).

The output of this driver can be used with various image conversion and manipulation utilities provided by NETPBM. Based on Jef Poskanzer's PBMPLUS package, NETPBM provides programs to convert the above PBM formats to GIF, TIFF, MacPaint, Macintosh PICT, PCX, X11 bitmap and many others. Complete information is available at http://netpbm.sourceforge.net/.

Examples:

```
set terminal pbm small monochrome                # defaults
set terminal pbm color medium size 800,600
set output '| pnmrotate 45 | pnm-
topng > tilted.png'  # uses NETPBM"
```

3.1.0.52 dospc

The ‘dospc‘ terminal driver supports PCs with arbitrary graphics boards, which will be automatically detected. It should be used only if you are not using the gcc or Zortec C/C++ compilers."

3.1.0.53 pdf

This terminal produces files in the Adobe Portable Document Format (PDF), useable for printing or display with tools like Acrobat Reader

Syntax:

```
      set terminal pdf {monochrome|color|colour}
                       {{no}enhanced}
                       {fname "<font>"} {fsize <fontsize>}
                       {linewidth <lw>} {rounded|butt}
                       {solid|dashed} {dl <dashlength>}}
```

The default is to use a different color for each line type. Selecting ‘monochome‘ will use black for all linetypes, in which case you probably want to select ‘dashed‘ to distinguish line types. Even in in mono mode you can still use explicit colors for filled areas or linestyles.

where `<font>` is the name of the default font to use (default Helvetica) and `<fontsize>` is the font size (in points, default 12). For help on which fonts are available or how to install new ones, please see the documentation for your local installation of pdflib.

The ‘enhanced‘ option enables enhanced text processing features (subscripts, superscripts and mixed fonts). See ‘enhanced‘.

The width of all lines in the plot can be increased by the factor `<n>` specified in ‘linewidth‘. Similarly ‘dashlength‘ is a multiplier for the default dash spacing.

‘rounded‘ sets line caps and line joins to be rounded; ‘butt‘ is the default, butt caps and mitered joins;"

* does not work.

3.1.0.54 pstricks

The ‘pstricks‘ driver is intended for use with the "pstricks.sty" macro package for LaTeX. It is an alternative to the ‘eepic‘ and ‘latex‘ drivers. You need "pstricks.sty", and, of course, a printer that understands PostScript, or a converter such as Ghostscript.

PSTricks is available via anonymous ftp from the /pub directory at Princeton.edu. This driver definitely does not come close to using the full capability of the PSTricks package.

Syntax:

```
      set terminal pstricks {hacktext | nohacktext} {unit | nounit}
```

The first option invokes an ugly hack that gives nicer numbers; the second has to do with plot scaling. The defaults are ‘hacktext‘ and ‘nounit‘."

3.1.0.55 qms

The ‘qms‘ terminal driver supports the QMS/QUIC Laser printer, the Talaris 1200 and others. It has no options."

3.1.0.56 regis

The ‘regis‘ terminal device generates output in the REGIS graphics language. It has the option of using 4 (the default) or 16 colors.

Syntax:

```
set terminal regis {4 | 16}"
```

3.1.0.57 regis

The ‘regis‘ terminal device generates output in the REGIS graphics language. It has the option of using 4 (the default) or 16 colors.

Syntax:

```
set terminal regis {4 | 16}"
```

3.1.0.58 rgip

The ‘rgip‘ and ‘uniplex‘ terminal drivers support RGIP metafiles. They can combine several graphs on a single page, but only one page is allowed in a given output file.

Syntax:

```
set terminal rgip | uniplex {portrait | landscape}
                            {[<horiz>,<vert>]} {<fontsize>}
```

permissible values for the font size are in the range 1–8, with the default being 1. The default layout is landscape. Graphs are placed on the page in a ‘horiz‘x‘vert‘ grid, which defaults to [1,1].

Example:

```
set terminal uniplex portrait [2,3]
```

puts six graphs on a page in three rows of two in portrait orientation."

3.1.0.59 sun

The ‘sun‘ terminal driver supports the SunView window system. It has no options."

3.1.0.60 svg

This terminal produces files in the W3C Scalable Vector Graphics format.

Syntax:

```
set terminal svg {size <x> <y> {|fixed|dynamic}}
                 {{no}enhanced}
                 {fname "<font>"} {fsize <fontsize>}
                 {fontfile <filename>}
                 {linewidth <lw>}
```

where <x> and <y> are the size of the SVG plot to generate, ‘dynamic‘ allows a svg-viewer to resize plot, whereas the default setting, ‘fixed‘, will request an absolute size.

‘linewidth <w>‘ increases the width of all lines used in the figure by a factor of <w>.

<font> is the name of the default font to use (default Arial) and <fontsize> is the font size (in points, default 12). Gnuplot does not currently provide a mechanism for embedding fonts in the output file, so svg viewing programs may substitute other fonts when the file is displayed.

The svg terminal supports an enhanced text mode, which allows font and other formatting commands to be embedded in labels and other text strings. The enhanced text mode syntax is shared with other gnuplot terminal types. See ‘enhanced‘ for more details.

SVG allows you to embed fonts directly into an SVG document, or to provide a hypertext link to the desired font. The ‘fontfile‘ option specifies a local file which is copied into the <defs> section of the resulting SVG output file. This file may either itself contain a font, or may contain the records necessary to create a hypertext reference to the desired font. Gnuplot will look for the requested file using the directory list in the GNUPLOT_FONTPATH environmental variable."

3.1.0.61 tek410x

The ‘tek410x‘ terminal driver supports the 410x and 420x family of Tektronix terminals. It has no options."

3.1.0.62 tek410x

The ‘tek410x‘ terminal driver supports the 410x and 420x family of Tektronix terminals. It has no options."

3.1.0.63 tek40

This family of terminal drivers supports a variety of VT-like terminals. ‘tek40xx‘ supports Tektronix 4010 and others as well as most TEK emulators; ‘vttek‘ supports VT-like tek40xx terminal emulators; ‘kc-tek40xx‘ supports MS-DOS Kermit Tek4010 terminal emulators in color: ‘km-tek40xx‘ supports them in monochrome; ‘selanar‘ supports Selanar graphics; and ‘bitgraph‘ supports BBN Bitgraph terminals. None have any options."

3.1.0.64 texdraw

The ‘texdraw‘ terminal driver supports the LaTeX texdraw environment. It is intended for use with "texdraw.sty" and "texdraw.tex" in the texdraw package.

Points, among other things, are drawn using the LaTeX commands "\\Diamond" and "\\Box". These commands no longer belong to the LaTeX2e core; they are included in the latexsym package, which is part of the base distribution and thus part of any LaTeX implementation. Please do not forget to use this package.

It has no options."

3.1.0.65 tgif

Tgif is an X11-based drawing tool—it has nothing to do with GIF.

The ‘tgif‘ driver supports different pointsizes (with Section 2.21.51 [pointsize], page 126), different label fonts and font sizes (e.g. ‘set label "Hallo" at x,y font "Helvetica,34"‘) and multiple graphs on the page. The proportions of the axes are not changed.

Syntax:

```
set terminal tgif {portrait | landscape | default} {<[x,y]>}
                  {monochrome | color}
                  {{linewidth | lw} <LW>}
                  {solid | dashed}
                  {font "<fontname>"} {<fontsize>}
```

where <[x,y]> specifies the number of graphs in the x and y directions on the page, ‘color‘ enables color, ‘linewidth‘ scales all linewidths by <LW>, "<fontname>" is the name of a valid PostScript font, and <fontsize> specifies the size of the PostScript font. ‘defaults‘ sets all options to their defaults: ‘portrait‘, ‘[1,1]‘, ‘color‘, ‘linwidth 1.0‘, ‘dashed‘, ‘"Helvetica"‘, and ‘18‘.

The ‘solid‘ option is usually prefered if lines are colored, as they often are in the editor. Hardcopy will be black-and-white, so ‘dashed‘ should be chosen for that.

Multiplot is implemented in two different ways.

The first multiplot implementation is the standard gnuplot multiplot feature:

```
set terminal tgif
set output "file.obj"
set multiplot
set origin x01,y01
set size  xs,ys
plot ...
     ...
set origin x02,y02
plot ...
unset multiplot
```

See Section 2.21.38 [multiplot], page 111 for further information.

The second version is the [x,y] option for the driver itself. The advantage of this implementation is that everything is scaled and placed automatically without the need for setting origins and sizes; the graphs keep their natural x/y proportions of 3/2 (or whatever is fixed by Section 2.21.57 [size], page 129).

If both multiplot methods are selected, the standard method is chosen and a warning message is given.

Examples of single plots (or standard multiplot):

```
set terminal tgif                  # defaults
set terminal tgif "Times-Roman" 24
set terminal tgif landscape
set terminal tgif landscape solid
```

Examples using the built-in multiplot mechanism:

```
set terminal tgif portrait [2,4]  # portrait; 2 plots in the x-
                                  # and 4 in the y-direction
set terminal tgif [1,2]           # portrait; 1 plot in the x-
                                  # and 2 in the y-direction
set terminal tgif landscape [3,3] # landscape; 3 plots in both
                                  # directions"
```

3.1.0.66 tgif

Tgif is an X11-based drawing tool—it has nothing to do with GIF.

The 'tgif' driver supports different pointsizes (with Section 2.21.51 [pointsize], page 126), different label fonts and font sizes (e.g. 'set label "Hallo" at x,y font "Helvetica,34"') and multiple graphs on the page. The proportions of the axes are not changed.

Syntax:

```
set terminal tgif {portrait | landscape | default} {<[x,y]>}
                  {monochrome | color}
                  {{linewidth | lw} <LW>}
                  {solid | dashed}
                  {font "<fontname>"} {<fontsize>}
```

where <[x,y]> specifies the number of graphs in the x and y directions on the page, 'color' enables color, 'linewidth' scales all linewidths by <LW>, "<fontname>" is the name of a valid PostScript font, and <fontsize> specifies the size of the PostScript font. 'defaults' sets all options to their defaults: 'portrait', '[1,1]', 'color', 'linwidth 1.0', 'dashed', '"Helvetica"', and '18'.

The 'solid' option is usually prefered if lines are colored, as they often are in the editor. Hardcopy will be black-and-white, so 'dashed' should be chosen for that.

Multiplot is implemented in two different ways.

The first multiplot implementation is the standard gnuplot multiplot feature:

```
set terminal tgif
set output "file.obj"
set multiplot
set origin x01,y01
set size  xs,ys
plot ...
     ...
set origin x02,y02
plot ...
unset multiplot
```

See Section 2.21.38 [multiplot], page 111 for further information.

The second version is the [x,y] option for the driver itself. The advantage of this implementation is that everything is scaled and placed automatically without the need for setting origins and sizes; the graphs keep their natural x/y proportions of 3/2 (or whatever is fixed by Section 2.21.57 [size], page 129).

If both multiplot methods are selected, the standard method is chosen and a warning message is given.

Examples of single plots (or standard multiplot):

```
set terminal tgif                  # defaults
set terminal tgif "Times-Roman" 24
set terminal tgif landscape
set terminal tgif landscape solid
```

Examples using the built-in multiplot mechanism:

```
set terminal tgif portrait [2,4]  # portrait; 2 plots in the x-
                                  # and 4 in the y-direction
set terminal tgif [1,2]           # portrait; 1 plot in the x-
                                  # and 2 in the y-direction
set terminal tgif landscape [3,3] # landscape; 3 plots in both
                                  # directions"
```

3.1.0.67 tkcanvas

This terminal driver generates Tk canvas widget commands based on Tcl/Tk (default) or Perl. To use it, rebuild 'gnuplot' (after uncommenting or inserting the appropriate line in "term.h"), then

```
gnuplot> set term tkcanvas {perltk} {interactive}
gnuplot> set output 'plot.file'
```

After invoking "wish", execute the following sequence of Tcl/Tk commands:

```
% source plot.file
% canvas .c
% pack .c
% gnuplot .c
```

Or, for Perl/Tk use a program like this:

```
use Tk;
my $top = MainWindow->new;
my $c = $top->Canvas->pack;
my $gnuplot = do "plot.pl";
$gnuplot->($c);
MainLoop;
```

The code generated by 'gnuplot' creates a procedure called "gnuplot" that takes the name of a canvas as its argument. When the procedure is called, it clears the canvas, finds the size of the canvas and draws the plot in it, scaled to fit.

For 2-dimensional plotting ('plot') two additional procedures are defined: "gnuplot_plotarea" will return a list containing the borders of the plotting area "xleft, xright, ytop, ybot" in canvas screen coordinates, while the ranges of the two axes "x1min, x1max, y1min, y1max, x2min, x2max, y2min, y2max" in plot coordinates can be obtained calling "gnuplot_axisranges". If the "interactive" option is specified, mouse clicking on a line segment will print the coordinates of its midpoint to stdout. Advanced actions can happen instead if the user supplies a procedure named "user_gnuplot_coordinates", which takes the following arguments: "win id x1s y1s x2s y2s x1e y1e x2e y2e x1m y1m x2m y2m", the name of the canvas and the id of the line segment followed by the coordinates of its start and end point in the two possible axis ranges; the coordinates of the midpoint are only filled for logarithmic axes.

The current version of ‘tkcanvas‘ supports neither Section 2.21.38 [multiplot], page 111 nor Section 2.17 [replot], page 71."

3.1.0.68 tpic

The ‘tpic‘ terminal driver supports the LaTeX picture environment with tpic \\specials. It is an alternative to the ‘latex‘ and ‘eepic‘ terminal drivers. Options are the point size, line width, and dot-dash interval.

Syntax:

```
set terminal tpic <pointsize> <linewidth> <interval>
```

where Section 2.21.51 [pointsize], page 126 and ‘linewidth‘ are integers in milli-inches and ‘interval‘ is a float in inches. If a non-positive value is specified, the default is chosen: pointsize = 40, linewidth = 6, interval = 0.1.

All drivers for LaTeX offer a special way of controlling text positioning: If any text string begins with '{', you also need to include a '}' at the end of the text, and the whole text will be centered both horizontally and vertically by LaTeX. — If the text string begins with '[', you need to continue it with: a position specification (up to two out of t,b,l,r), ']{', the text itself, and finally, '}'. The text itself may be anything LaTeX can typeset as an LR-box. \\rule{}{}'s may help for best positioning.

Examples: About label positioning: Use gnuplot defaults (mostly sensible, but sometimes not really best):

```
set title '\\LaTeX\\ -- $ \\gamma $'
```

Force centering both horizontally and vertically:

```
set label '{\\LaTeX\\ -- $ \\gamma $}' at 0,0
```

Specify own positioning (top here):

```
set xlabel '[t]{\\LaTeX\\ -- $ \\gamma $}'
```

The other label – account for long ticlabels:

```
set ylabel '[r]{\\LaTeX\\ -- $ \\gamma $\\rule{7mm}{0pt}}'"
```

3.1.0.69 unixpc

The ‘unixpc‘ terminal driver supports AT&T 3b1 and AT&T 7300 Unix PC. It has no options."

3.1.0.70 unixplot

The ‘unixplot‘ terminal driver generates output in the Unix "plot" graphics language. It has no options.

This terminal cannot be compiled if the GNU version of plot is to be used; in that case, use the ‘gnugraph‘ terminal instead."

3.1.0.71 vx384

The ‘vx384‘ terminal driver supports the Vectrix 384 and Tandy color printers. It has no options."

3.1.0.72 vgagl

The ‘vgagl‘ driver is a fast linux console driver with full mouse and pm3d support. It looks at the environment variable SVGALIB_DEFAULT_MODE for the default mode; if not set, it uses a 256 color mode with the highest available resolution.

Syntax:

```
set terminal vgagl \\
             background [red] [[green] [blue]] \\
             [uniform | interpolate] \\
             [dump "file"] \\
             [mode]
```

The color mode can also be given with the mode option. Both Symbolic names as G1024x768x256 and integers are allowed. The ‘background‘ option takes either one or three integers in the range [0, 255]. If only one integers is supplied, it is taken as gray value for the background. If three integers are present, the background gets the corresponding color. The (mutually exclusive) options ‘interpolate‘ and ‘uniform‘ control if color interpolation is done while drawing triangles (on by default).

A Section 2.21.50.4 [file], page 125 can be specified with the ‘dump "file"‘ option. If this option is present, (i.e the dump file name is not empty) pressing the key KP_Delete will write the file. This action cannot and cannot be rebound. The file is written in raw ppm (P6) format. Note that this option is reset each time the ‘set term‘ command is issued.

To get high resolution modes, you will probably have to modify the configuration file of libvga, usually /etc/vga/libvga.conf. Using the VESA fb is a good choice, but this needs to be compiled in the kernel.

The vgagl driver uses the first *available* vga mode from the following list:

```
  - the driver which was supplied when set-
 ting vgagl, e.g. ‘set term vgagl
    G1024x768x256‘ would first check, if the G1024x768x256 mode is available.
  - the environment variable SVGALIB_DEFAULT_MODE
  - G1024x768x256
  - G800x600x256
  - G640x480x256
  - G320x200x256
  - G1280x1024x256
  - G1152x864x256
  - G1360x768x256
  - G1600x1200x256
```

3.1.0.73 VWS

The ‘VWS‘ terminal driver supports the VAX Windowing System. It has no options. It will sense the display type (monochrome, gray scale, or color.) All line styles are plotted as solid lines."

3.1.0.74 windows

Three options may be set in the 'windows' terminal driver.

Syntax:

```
set terminal windows {<color>} {"<fontname>"} {<fontsize>}
```

where '`<color>`' is either 'color' or 'monochrome', '`"<fontname>"`' is the name of a valid Windows font, and '`<fontsize>`' is the size of the font in points.

Other options may be set with the graph-menu, the initialization file, and 'set linestyle'. Note that there is one restriction imposed by the classic Windows GDI interface: modifiable linewidth only works with solid lines, not with dotted or dashed ones. /* Does this really belong here? If not, someone move it where it does. */

The Windows version normally terminates immediately as soon as the end of any files given as command line arguments is reached (i.e. in non-interactive mode), unless you specify '-' as the last command line option. It will also not show the text-window at all, in this mode, only the plot. By giving the optional argument '-persist' (same as for gnuplot under x11; former Windows-only options '/noend' or '-noend' are still accepted as well), will not close gnuplot. Contrary to gnuplot on other operating systems, gnuplot's interactive command line is accessible after the -persist option.

— GRAPH-MENU —

The 'gnuplot graph' window has the following options on a pop-up menu accessed by pressing the right mouse button or selecting 'Options' from the system menu:

'Bring to Top' when checked brings the graph window to the top after every plot.

'Color' when checked enables color linestyles. When unchecked it forces monochrome linestyles.

'Copy to Clipboard' copies a bitmap and a Metafile picture.

'Background...' sets the window background color.

'Choose Font...' selects the font used in the graphics window.

'Line Styles...' allows customization of the line colors and styles.

'Print...' prints the graphics windows using a Windows printer driver and allows selection of the printer and scaling of the output. The output produced by 'Print' is not as good as that from 'gnuplot''s own printer drivers.

'Update wgnuplot.ini' saves the current window locations, window sizes, text window font, text window font size, graph window font, graph window font size, background color and linestyles to the initialization file 'WGNUPLOT.INI'.

— PRINTING —

In order of preference, graphs may be be printed in the following ways.

'1.' Use the 'gnuplot' command 'set terminal' to select a printer and Section 2.21.46 [output], page 115 to redirect output to a file.

'2.' Select the 'Print...' command from the 'gnuplot graph' window. An extra command 'screendump' does this from the text window.

'3.' If 'set output `"PRN"`' is used, output will go to a temporary file. When you exit from 'gnuplot' or when you change the output with another Section 2.21.46 [output], page 115

command, a dialog box will appear for you to select a printer port. If you choose OK, the output will be printed on the selected port, passing unmodified through the print manager. It is possible to accidentally (or deliberately) send printer output meant for one printer to an incompatible printer.

— TEXT-MENU —

The ‘gnuplot text‘ window has the following options on a pop-up menu accessed by pressing the right mouse button or selecting ‘Options‘ from the system menu:

‘Copy to Clipboard‘ copies marked text to the clipboard.

‘Paste‘ copies text from the clipboard as if typed by the user.

‘Choose Font...‘ selects the font used in the text window.

‘System Colors‘ when selected makes the text window honor the System Colors set using the Control Panel. When unselected, text is black or blue on a white background.

‘Update wgnuplot.ini‘ saves the current text window location, text window size, text window font and text window font size to the initialisation file ‘WGNUPLOT.INI‘.

‘MENU BAR‘

If the menu file ‘WGNUPLOT.MNU‘ is found in the same directory as WGNUPLOT.EXE, then the menu specified in ‘WGNUPLOT.MNU‘ will be loaded. Menu commands:

[Menu] starts a new menu with the name on the following line.

[EndMenu] ends the current menu.

[–] inserts a horizontal menu separator.

[|] inserts a vertical menu separator.

[Button] puts the next macro on a push button instead of a menu.

Macros take two lines with the macro name (menu entry) on the first line and the macro on the second line. Leading spaces are ignored. Macro commands:

[INPUT] — Input string with prompt terminated by [EOS] or {ENTER}

[EOS] — End Of String terminator. Generates no output.

[OPEN] — Get name of file to open from list box, with title of list box terminated by [EOS], followed by default filename terminated by [EOS] or {ENTER}. This uses COMMDLG.DLL from Windows 3.1.

[SAVE] — Get name of file to save. Similar to [OPEN]

Macro character substitutions:

{ENTER} — Carriage Return '\\r'

{TAB} — Tab '\\011'

{ESC} — Escape '\\033'

{^A} — '\\001'

...

{^_} — '\\031'

Macros are limited to 256 characters after expansion.

— WGNUPLOT.INI —

Windows ‘gnuplot‘ will read some of its options from the ‘[WGNUPLOT]‘ section of ‘WGNUPLOT.INI‘ in the Windows directory. A sample ‘WGNUPLOT.INI‘ file:

```
[WGNUPLOT]
TextOrigin=0 0
TextSize=640 150
TextFont=Terminal,9
GraphOrigin=0 150
GraphSize=640 330
GraphFont=Arial,10
GraphColor=1
GraphToTop=1
GraphBackground=255 255 255
Border=0 0 0 0 0
Axis=192 192 192 2 2
Line1=0 0 255 0 0
Line2=0 255 0 0 1
Line3=255 0 0 0 2
Line4=255 0 255 0 3
Line5=0 0 128 0 4
```

The ‘GraphFont‘ entry specifies the font name and size in points. The five numbers given in the ‘Border‘, ‘Axis‘ and ‘Line‘ entries are the ‘Red‘ intensity (0–255), ‘Green‘ intensity, ‘Blue‘ intensity, ‘Color Linestyle‘ and ‘Mono Linestyle‘. ‘Linestyles‘ are 0=SOLID, 1=DASH, 2=DOT, 3=DASHDOT, 4=DASHDOTDOT. In the sample ‘WGNUPLOT.INI‘ file above, Line 2 is a green solid line in color mode, or a dashed line in monochrome mode. The default line width is 1 pixel. If ‘Linestyle‘ is negative, it specifies the width of a SOLID line in pixels. Line1 and any linestyle used with the ‘points‘ style must be SOLID with unit width.

— WINDOWS3.0 —

Windows 3.1 is preferred, but WGNUPLOT will run under Windows 3.0 with the following restrictions: ‘1.‘ COMMDLG.DLL and SHELL.DLL (available with Windows 3.1 or Borland C++ 3.1) must be in the windows directory.

‘2.‘ WGNUPLOT.HLP produced by Borland C++ 3.1 is in Windows 3.1 format. You need to use the WINHELP.EXE supplied with Borland C++ 3.1.

‘3.‘ It will not run in real mode due to lack of memory.

‘4.‘ TrueType fonts are not available in the graph window.

‘5.‘ Drag-drop does not work."

3.1.0.75 x11

‘gnuplot‘ provides the ‘x11‘ terminal type for use with X servers. This terminal type is set automatically at startup if the ‘DISPLAY‘ environment variable is set, if the ‘TERM‘ environment variable is set to ‘xterm‘, or if the ‘-display‘ command line option is used.

Syntax:

```
set terminal x11 {<n>}
                 {title "<string>"}
                 {{no}enhanced}
                 {font <fontspec>}
```

```
                    {{no}persist} {{no}raise}
                    {close}
set terminal x11 {reset}
```

Multiple plot windows are supported: ‘set terminal x11 <n>‘ directs the output to plot window number n. If n is not 0, the terminal number will be appended to the window title (unless a title has been supplied manually) and the icon will be labeled ‘Gnuplot <n>‘. The active window may be distinguished by a change in cursor (from default to crosshair).

The x11 terminal support enhanced text mode (see ‘enhanced‘), subject to the available fonts. In order for font size commands embedded in text to have any effect, the default x11 font must be scalable. Thus the first example below will work as expected, but the second will not.

```
set term x11 enhanced font "arial,15"
set title '{/=20 Big} Medium {/=5 Small}'

set term x11 enhanced font "terminal-14"
set title '{/=20 Big} Medium {/=5 Small}'
```

Plot windows remain open even when the ‘gnuplot‘ driver is changed to a different device. A plot window can be closed by pressing the letter q while that window has input focus, or by choosing ‘close‘ from a window manager menu. All plot windows can be closed by specifying Section 2.19 [reset], page 73, which actually terminates the subprocess which maintains the windows (unless ‘-persist‘ was specified). The ‘close‘ command can be used to close individual plot windows by number. However, after a Section 2.19 [reset], page 73, those plot windows left due to persist cannot be closed with the command ‘close‘. A ‘close‘ without a number closes the current active plot window.

The gnuplot outboard driver, gnuplot_x11, is searched in a default place chosen when the program is compiled. You can override that by defining the environment variable GNUPLOT_DRIVER_DIR to point to a different location.

Plot windows will automatically be closed at the end of the session unless the ‘-persist‘ option was given.

The options ‘persist‘ and Section 2.16 [raise], page 71 are unset by default, which means that the defaults (persist == no and raise == yes) or the command line options -persist / -raise or the Xresources are taken. If [no]persist or [no]raise are specified, they will override command line options and Xresources. Setting one of these options takes place immediately, so the behaviour of an already running driver can be modified.

The option ‘title `"<title name>"`‘ will supply the title name of the window for the current plot window or plot window `<n>` if a number is given. Where (or if) this title is shown depends on your X window manager.

The size or aspect ratio of a plot may be changed by resizing the ‘gnuplot‘ window.

Linewidths and pointsizes may be changed from within ‘gnuplot‘ with ‘set linestyle‘.

For terminal type ‘x11‘, ‘gnuplot‘ accepts (when initialized) the standard X Toolkit options and resources such as geometry, font, and name from the command line arguments or a configuration file. See the X(1) man page (or its equivalent) for a description of such options.

A number of other 'gnuplot' options are available for the 'x11' terminal. These may be specified either as command-line options when 'gnuplot' is invoked or as resources in the configuration file ".Xdefaults". They are set upon initialization and cannot be altered during a 'gnuplot' session. (except 'persist' and Section 2.16 [raise], page 71)

— X11_FONTS —

Upon initial startup, the default font is taken from the X11 resources as set in the system or user .Xdefaults file or on the command line.

Example:

```
gnuplot*font: lucidasans-bold-12
```

A new default font may be specified to the x11 driver from inside gnuplot using

```
'set term x11 font "<fontspec>"'
```

The driver first queries the X-server for a font of the exact name given. If this query fails, then it tries to interpret <fontspec> as "<font>,<size>,<slant>,<weight>" and to construct a full X11 font name of the form

```
-*-<font>-<weight>-<s>-*-*-<size>-*-*-*-*-*-<encoding>


 <font> is the base name of the font (e.g. Times or Symbol)
 <size> is the point size (defaults to 12 if not specified)
 <s> is 'i' if <slant>=="italic" 'o' if <slant>=="oblique" 'r' otherwise
 <weight> is 'medium' or 'bold' if explicitly requested, otherwise '*'
 <encoding> is set based on the current character set (see Sec-
tion 2.21.18 [encoding], page 91).
```

So 'set term x11 font "arial,15,italic"' will be translated to -*-arial-*-i-*-*-15-*-*-*-*-*-iso8859-1 (assuming default encoding). The <size>, <slant>, and <weight> specifications are all optional. If you do not specify <slant> or <weight> then you will get whatever font variant the font server offers first. The driver also recognizes some common PostScript font names and replaces them with possible X11 or TrueType equivalents. This same sequence is used to process font requests from Section 2.21.29 [label], page 104.

If your gnuplot was built with configuration option –enable-x11-mbfonts, you can specify multi-byte fonts by using the prefix "mbfont:" on the font name. An additional font may be given, separated by a semicolon. Since multi-byte font encodings are interpreted according to the locale setting, you must make sure that the environmental variable LC_CTYPE is set to some appropriate locale value such as ja_JP.eucJP, ko_KR.EUC, or zh_CN.EUC.

Example:

```
set term x11 font 'mbfont:kana14;k14'
      # 'kana14' and 'k14' are Japanese X11 font aliases, and ';'
      # is the separator of font names.
set term x11 font 'mbfont:fixed,16,r,medium'
      # <font>,<size>,<slant>,<weight> form is also usable.
set title '(mb strings)' font 'mbfont:*-fixed-medium-r-normal--
14-*'
```

The same syntax applies to the default font in Xresources settings, for example,

```
        gnuplot*font: \\
            mbfont:-misc-fixed-medium-r-normal--14-*-*-*-c-*-
  jisx0208.1983-0
```

If gnuplot is built with –enable-x11-mbfonts, you can use two special PostScript font names 'Ryumin-Light-*' and 'GothicBBB-Medium-*' (standard Japanese PS fonts) without the prefix "mbfont:".

— COMMAND-LINE_OPTIONS —

In addition to the X Toolkit options, the following options may be specified on the command line when starting 'gnuplot' or as resources in your ".Xdefaults" file (note that Section 2.16 [raise], page 71 and 'persist' can be overridden later by 'set term x11 [no]raise [no]persist)':

```
 '-mono'     forces monochrome rendering on color displays.
 '-gray'     requests grayscale render-
ing on grayscale or color displays.
             (Grayscale displays receive monochrome render-
ing by default.)
 '-clear'    requests that the window be cleared momentarily before a
             new plot is displayed.
 '-tvtwm'    requests that geometry specifications for position of the
             window be made relative to the currently dis-
played portion
             of the virtual root.
 '-raise'    raises plot window after each plot
 '-noraise'  does not raise plot window after each plot
 '-noevents' does not process mouse and key events
 '-ctrlq   ' closes window on ctrl-q rather than q
 '-persist'  plot windows survive after main gnuplot program exits
```

The options are shown above in their command-line syntax. When entered as resources in ".Xdefaults", they require a different syntax.

Example:

```
        gnuplot*gray:  on
        gnuplot*ctrlq: on
```

'gnuplot' also provides a command line option ('-pointsize <v>') and a resource, 'gnuplot*pointsize: <v>', to control the size of points plotted with the 'points' plotting style. The value 'v' is a real number (greater than 0 and less than or equal to ten) used as a scaling factor for point sizes. For example, '-pointsize 2' uses points twice the default size, and '-pointsize 0.5' uses points half the normal size.

The '-noevents' switch disables all mouse and key event processing (except for 'q' and '<space>' for closing the window). This is useful for programs which use the x11 driver independent of the gnuplot main program.

The ‘-ctrlq‘ switch changes the hot-key that closes a plot window from ‘q‘ to ‘<ctrl>q‘. This is useful is you are using the keystroke-capture feature ‘pause mouse keystroke‘, since it allows the character ‘q‘ to be captured just as all other alphanumeric characters. The ‘-ctrlq‘ switch similarly replaces the <space> hot-key with <ctrl><space> for the same reason.

— MONOCHROME_OPTIONS —

For monochrome displays, ‘gnuplot‘ does not honor foreground or background colors. The default is black-on-white. ‘-rv‘ or ‘gnuplot*reverseVideo: on‘ requests white-on-black.

— COLOR_RESOURCES —

For color displays, ‘gnuplot‘ honors the following resources (shown here with their default values) or the greyscale resources. The values may be color names as listed in the X11 rgb.txt file on your system, hexadecimal RGB color specifications (see X11 documentation), or a color name followed by a comma and an ‘intensity‘ value from 0 to 1. For example, ‘blue, 0.5‘ means a half intensity blue.

```
gnuplot*background:  white
gnuplot*textColor:   black
gnuplot*borderColor: black
gnuplot*axisColor:   black
gnuplot*line1Color:  red
gnuplot*line2Color:  green
gnuplot*line3Color:  blue
gnuplot*line4Color:  magenta
gnuplot*line5Color:  cyan
gnuplot*line6Color:  sienna
gnuplot*line7Color:  orange
gnuplot*line8Color:  coral
```

The command-line syntax for these is simple only for background, which maps directly to the usual X11 toolkit option "-bg". All others can only be set on the command line by use of the generic "-xrm" resource override option

Examples:

```
gnuplot -background coral
```

to change the background color.

```
gnuplot -xrm 'gnuplot*line1Color:blue'
```

to override the first linetype color.

— GRAYSCALE_RESOURCES —

When ‘-gray‘ is selected, ‘gnuplot‘ honors the following resources for grayscale or color displays (shown here with their default values). Note that the default background is black.

```
gnuplot*background: black
gnuplot*textGray:   white
gnuplot*borderGray: gray50
gnuplot*axisGray:   gray50
gnuplot*line1Gray:  gray100
gnuplot*line2Gray:  gray60
gnuplot*line3Gray:  gray80
```

```
gnuplot*line4Gray:  gray40
gnuplot*line5Gray:  gray90
gnuplot*line6Gray:  gray50
gnuplot*line7Gray:  gray70
gnuplot*line8Gray:  gray30
```

— LINE_RESOURCES —

'gnuplot' honors the following resources for setting the width (in pixels) of plot lines (shown here with their default values.) 0 or 1 means a minimal width line of 1 pixel width. A value of 2 or 3 may improve the appearance of some plots.

```
gnuplot*borderWidth: 2
gnuplot*axisWidth:   0
gnuplot*line1Width:  0
gnuplot*line2Width:  0
gnuplot*line3Width:  0
gnuplot*line4Width:  0
gnuplot*line5Width:  0
gnuplot*line6Width:  0
gnuplot*line7Width:  0
gnuplot*line8Width:  0
```

'gnuplot' honors the following resources for setting the dash style used for plotting lines. 0 means a solid line. A two-digit number 'jk' ('j' and 'k' are >= 1 and <= 9) means a dashed line with a repeated pattern of 'j' pixels on followed by 'k' pixels off. For example, '16' is a dotted line with one pixel on followed by six pixels off. More elaborate on/off patterns can be specified with a four-digit value. For example, '4441' is four on, four off, four on, one off. The default values shown below are for monochrome displays or monochrome rendering on color or grayscale displays. For color displays, the default for each is 0 (solid line) except for 'axisDashes' which defaults to a '16' dotted line.

```
gnuplot*borderDashes:   0
gnuplot*axisDashes:    16
gnuplot*line1Dashes:    0
gnuplot*line2Dashes:   42
gnuplot*line3Dashes:   13
gnuplot*line4Dashes:   44
gnuplot*line5Dashes:   15
gnuplot*line6Dashes: 4441
gnuplot*line7Dashes:   42
gnuplot*line8Dashes:   13
```

, "

— X11 PM3D_RESOURCES —

Choosing the appropriate visual class and number of colors is a crucial point in X11 applications and a bit awkward, since X11 supports six visual types in different depths.

By default ‘gnuplot‘ uses the default visual of the screen. The number of colors which can be allocated depends on the visual class chosen. On a visual class with a depth > 12bit, gnuplot starts with a maximal number of 0x200 colors. On a visual class with a depth > 8bit (but <= 12 bit) the maximal number of colors is 0x100, on <= 8bit displays the maximum number of colors is 240 (16 are left for line colors).

Gnuplot first starts to allocate the maximal number of colors as stated above. If this fails, the number of colors is reduced by the factor 2 until gnuplot gets all colors which are requested. If dividing ‘maxcolors‘ by 2 repeatedly results in a number which is smaller than ‘mincolors‘ ‘gnuplot‘ tries to install a private colormap. In this case the window manager is responsible for swapping colormaps when the pointer is moved in and out the x11 driver's window.

The default for ‘mincolors‘ is maxcolors / (num_colormaps > 1 ? 2 : 8), where num_colormaps is the number of colormaps which are currently used by gnuplot (usually 1, if only one x11 window is open).

Some systems support multiple (different) visual classes together on one screen. On these systems it might be necessary to force gnuplot to use a specific visual class, e.g. the default visual might be 8bit PseudoColor but the screen would also support 24bit TrueColor which would be the preferred choice.

The information about an Xserver's capabilities can be obtained with the program ‘xdpyinfo‘. For the visual names below you can choose one of StaticGray, GrayScale, StaticColor, PseudoColor, TrueColor, DirectColor. If an Xserver supports a requested visual type at different depths, ‘gnuplot‘ chooses the visual class with the highest depth (deepest). If the requested visual class matches the default visual and multiple classes of this type are supported, the default visual is preferred.

Example: on an 8bit PseudoColor visual you can force a private color map by specifying ‘gnuplot*maxcolors: 240‘ and ‘gnuplot*mincolors: 240‘.

```
gnuplot*maxcolors:  <integer>
gnuplot*mincolors:  <integer>
gnuplot*visual:     <visual name>
```

, "

— X11 OTHER_RESOURCES —

By default the contents of the current plot window are exported to the X11 clipboard in response to X events in the window. Setting the resource 'gnuplot*exportselection' to 'off' or 'false' will disable this.

By default text rotation is done using a method that is fast, but can corrupt nearby colors depending on the background. If this is a problem, you can set the resource 'gnuplot.fastrotate' to 'off'

```
gnuplot*exportselection:  off
gnuplot*fastrotate:  on
gnuplot*ctrlq:  off
```

3.1.0.76 x11

'gnuplot' provides the 'x11' terminal type for use with X servers. This terminal type is set automatically at startup if the 'DISPLAY' environment variable is set, if the 'TERM' environment variable is set to 'xterm', or if the '-display' command line option is used.

Syntax:

```
set terminal x11 {<n>}
                 {title "<string>"}
                 {{no}enhanced}
                 {font <fontspec>}
                 {{no}persist} {{no}raise}
                 {close}
set terminal x11 {reset}
```

Multiple plot windows are supported: 'set terminal x11 <n>' directs the output to plot window number n. If n is not 0, the terminal number will be appended to the window title (unless a title has been supplied manually) and the icon will be labeled 'Gnuplot <n>'. The active window may be distinguished by a change in cursor (from default to crosshair).

The x11 terminal support enhanced text mode (see 'enhanced'), subject to the available fonts. In order for font size commands embedded in text to have any effect, the default x11 font must be scalable. Thus the first example below will work as expected, but the second will not.

```
set term x11 enhanced font "arial,15"
set title '{/=20 Big} Medium {/=5 Small}'

set term x11 enhanced font "terminal-14"
set title '{/=20 Big} Medium {/=5 Small}'
```

Plot windows remain open even when the 'gnuplot' driver is changed to a different device. A plot window can be closed by pressing the letter q while that window has input focus, or by choosing 'close' from a window manager menu. All plot windows can be closed by specifying Section 2.19 [reset], page 73, which actually terminates the subprocess which maintains the windows (unless '-persist' was specified). The 'close' command can be used to close individual plot windows by number. However, after a Section 2.19 [reset], page 73, those plot windows left due to persist cannot be closed with the command 'close'. A 'close' without a number closes the current active plot window.

The gnuplot outboard driver, gnuplot_x11, is searched in a default place chosen when the program is compiled. You can override that by defining the environment variable GNUPLOT_DRIVER_DIR to point to a different location.

Plot windows will automatically be closed at the end of the session unless the '-persist' option was given.

The options 'persist' and Section 2.16 [raise], page 71 are unset by default, which means that the defaults (persist == no and raise == yes) or the command line options -persist / -raise or the Xresources are taken. If [no]persist or [no]raise are specified, they will override command line options and Xresources. Setting one of these options takes place immediately, so the behaviour of an already running driver can be modified.

The option ‘title "<title name>"‘ will supply the title name of the window for the current plot window or plot window <n> if a number is given. Where (or if) this title is shown depends on your X window manager.

The size or aspect ratio of a plot may be changed by resizing the ‘gnuplot‘ window.

Linewidths and pointsizes may be changed from within ‘gnuplot‘ with ‘set linestyle‘.

For terminal type ‘x11‘, ‘gnuplot‘ accepts (when initialized) the standard X Toolkit options and resources such as geometry, font, and name from the command line arguments or a configuration file. See the X(1) man page (or its equivalent) for a description of such options.

A number of other ‘gnuplot‘ options are available for the ‘x11‘ terminal. These may be specified either as command-line options when ‘gnuplot‘ is invoked or as resources in the configuration file ".Xdefaults". They are set upon initialization and cannot be altered during a ‘gnuplot‘ session. (except ‘persist‘ and Section 2.16 [raise], page 71)

— X11_FONTS —

Upon initial startup, the default font is taken from the X11 resources as set in the system or user .Xdefaults file or on the command line.

Example:

```
gnuplot*font: lucidasans-bold-12
```

A new default font may be specified to the x11 driver from inside gnuplot using

```
‘set term x11 font "<fontspec>"‘
```

The driver first queries the X-server for a font of the exact name given. If this query fails, then it tries to interpret <fontspec> as "<font>,<size>,<slant>,<weight>" and to construct a full X11 font name of the form

```
-*-<font>-<weight>-<s>-*-*-<size>-*-*-*-*-*-<encoding>
```

```
<font> is the base name of the font (e.g. Times or Symbol)
<size> is the point size (defaults to 12 if not specified)
<s> is ‘i‘ if <slant>=="italic" ‘o‘ if <slant>=="oblique" ‘r‘ otherwise
<weight> is ‘medium‘ or ‘bold‘ if explicitly requested, otherwise ‘*‘
<encoding> is set based on the current character set (see Section 2.21.18 [encoding], page 91).
```

So ‘set term x11 font "arial,15,italic"‘ will be translated to -*-arial-*-i-*-*-15-*-*-*-*-*-iso8859-1 (assuming default encoding). The <size>, <slant>, and <weight> specifications are all optional. If you do not specify <slant> or <weight> then you will get whatever font variant the font server offers first. The driver also recognizes some common PostScript font names and replaces them with possible X11 or TrueType equivalents. This same sequence is used to process font requests from Section 2.21.29 [label], page 104.

If your gnuplot was built with configuration option –enable-x11-mbfonts, you can specify multi-byte fonts by using the prefix "mbfont:" on the font name. An additional font may be given, separated by a semicolon. Since multi-byte font encodings are interpreted according to the locale setting, you must make sure that the environmental variable LC_CTYPE is set to some appropriate locale value such as ja_JP.eucJP, ko_KR.EUC, or zh_CN.EUC.

Example:

```
        set term x11 font 'mbfont:kana14;k14'
              # 'kana14' and 'k14' are Japanese X11 font aliases, and ';'
              # is the separator of font names.
        set term x11 font 'mbfont:fixed,16,r,medium'
              # <font>,<size>,<slant>,<weight> form is also usable.
        set title '(mb strings)' font 'mbfont:*-fixed-medium-r-normal--
  14-*'
```

The same syntax applies to the default font in Xresources settings, for example,

```
        gnuplot*font: \\
            mbfont:-misc-fixed-medium-r-normal--14-*-*-*-c-*-
  jisx0208.1983-0
```

If gnuplot is built with –enable-x11-mbfonts, you can use two special PostScript font names 'Ryumin-Light-*' and 'GothicBBB-Medium-*' (standard Japanese PS fonts) without the prefix "mbfont:".

— COMMAND-LINE_OPTIONS —

In addition to the X Toolkit options, the following options may be specified on the command line when starting 'gnuplot' or as resources in your ".Xdefaults" file (note that Section 2.16 [raise], page 71 and 'persist' can be overridden later by 'set term x11 [no]raise [no]persist)':

```
   '-mono'     forces monochrome rendering on color displays.
   '-gray'     requests grayscale render-
  ing on grayscale or color displays.
               (Grayscale displays receive monochrome render-
  ing by default.)
   '-clear'    requests that the window be cleared momentarily before a
               new plot is displayed.
   '-tvtwm'    requests that geometry specifications for position of the
               window be made relative to the currently dis-
  played portion
               of the virtual root.
   '-raise'    raises plot window after each plot
   '-noraise'  does not raise plot window after each plot
   '-noevents' does not process mouse and key events
   '-ctrlq   ' closes window on ctrl-q rather than q
   '-persist'  plot windows survive after main gnuplot program exits
```

The options are shown above in their command-line syntax. When entered as resources in ".Xdefaults", they require a different syntax.

Example:

```
        gnuplot*gray:  on
        gnuplot*ctrlq: on
```

'gnuplot' also provides a command line option ('-pointsize <v>') and a resource, 'gnuplot*pointsize: <v>', to control the size of points plotted with the 'points' plotting style. The value 'v' is a real number (greater than 0 and less than or equal to ten) used as a scaling factor for point sizes. For example, '-pointsize 2' uses points twice the default size, and '-pointsize 0.5' uses points half the normal size.

The '-noevents' switch disables all mouse and key event processing (except for 'q' and '<space>' for closing the window). This is useful for programs which use the x11 driver independent of the gnuplot main program.

The '-ctrlq' switch changes the hot-key that closes a plot window from 'q' to '<ctrl>q'. This is useful is you are using the keystroke-capture feature 'pause mouse keystroke', since it allows the character 'q' to be captured just as all other alphanumeric characters. The '-ctrlq' switch similarly replaces the <space> hot-key with <ctrl><space> for the same reason.

— MONOCHROME_OPTIONS —

For monochrome displays, 'gnuplot' does not honor foreground or background colors. The default is black-on-white. '-rv' or 'gnuplot*reverseVideo: on' requests white-on-black.

— COLOR_RESOURCES —

For color displays, 'gnuplot' honors the following resources (shown here with their default values) or the greyscale resources. The values may be color names as listed in the X11 rgb.txt file on your system, hexadecimal RGB color specifications (see X11 documentation), or a color name followed by a comma and an 'intensity' value from 0 to 1. For example, 'blue, 0.5' means a half intensity blue.

```
gnuplot*background:  white
gnuplot*textColor:   black
gnuplot*borderColor: black
gnuplot*axisColor:   black
gnuplot*line1Color:  red
gnuplot*line2Color:  green
gnuplot*line3Color:  blue
gnuplot*line4Color:  magenta
gnuplot*line5Color:  cyan
gnuplot*line6Color:  sienna
gnuplot*line7Color:  orange
gnuplot*line8Color:  coral
```

The command-line syntax for these is simple only for background, which maps directly to the usual X11 toolkit option "-bg". All others can only be set on the command line by use of the generic "-xrm" resource override option

Examples:

```
gnuplot -background coral
```

to change the background color.

```
gnuplot -xrm 'gnuplot*line1Color:blue'
```

to override the first linetype color.

— GRAYSCALE_RESOURCES —

When ‘-gray‘ is selected, ‘gnuplot‘ honors the following resources for grayscale or color displays (shown here with their default values). Note that the default background is black.

```
gnuplot*background: black
gnuplot*textGray:   white
gnuplot*borderGray: gray50
gnuplot*axisGray:   gray50
gnuplot*line1Gray:  gray100
gnuplot*line2Gray:  gray60
gnuplot*line3Gray:  gray80
gnuplot*line4Gray:  gray40
gnuplot*line5Gray:  gray90
gnuplot*line6Gray:  gray50
gnuplot*line7Gray:  gray70
gnuplot*line8Gray:  gray30
```

— LINE_RESOURCES —

‘gnuplot‘ honors the following resources for setting the width (in pixels) of plot lines (shown here with their default values.) 0 or 1 means a minimal width line of 1 pixel width. A value of 2 or 3 may improve the appearance of some plots.

```
gnuplot*borderWidth: 2
gnuplot*axisWidth:   0
gnuplot*line1Width:  0
gnuplot*line2Width:  0
gnuplot*line3Width:  0
gnuplot*line4Width:  0
gnuplot*line5Width:  0
gnuplot*line6Width:  0
gnuplot*line7Width:  0
gnuplot*line8Width:  0
```

‘gnuplot‘ honors the following resources for setting the dash style used for plotting lines. 0 means a solid line. A two-digit number ‘jk‘ (‘j‘ and ‘k‘ are >= 1 and <= 9) means a dashed line with a repeated pattern of ‘j‘ pixels on followed by ‘k‘ pixels off. For example, ’16’ is a dotted line with one pixel on followed by six pixels off. More elaborate on/off patterns can be specified with a four-digit value. For example, ’4441’ is four on, four off, four on, one off. The default values shown below are for monochrome displays or monochrome rendering on color or grayscale displays. For color displays, the default for each is 0 (solid line) except for ‘axisDashes‘ which defaults to a ’16’ dotted line.

```
gnuplot*borderDashes:   0
gnuplot*axisDashes:    16
gnuplot*line1Dashes:    0
gnuplot*line2Dashes:   42
gnuplot*line3Dashes:   13
gnuplot*line4Dashes:   44
gnuplot*line5Dashes:   15
```

```
gnuplot*line6Dashes: 4441
gnuplot*line7Dashes:   42
gnuplot*line8Dashes:   13
```

, "

— X11 PM3D_RESOURCES —

Choosing the appropriate visual class and number of colors is a crucial point in X11 applications and a bit awkward, since X11 supports six visual types in different depths.

By default 'gnuplot' uses the default visual of the screen. The number of colors which can be allocated depends on the visual class chosen. On a visual class with a depth > 12bit, gnuplot starts with a maximal number of 0x200 colors. On a visual class with a depth > 8bit (but <= 12 bit) the maximal number of colors is 0x100, on <= 8bit displays the maximum number of colors is 240 (16 are left for line colors).

Gnuplot first starts to allocate the maximal number of colors as stated above. If this fails, the number of colors is reduced by the factor 2 until gnuplot gets all colors which are requested. If dividing 'maxcolors' by 2 repeatedly results in a number which is smaller than 'mincolors' 'gnuplot' tries to install a private colormap. In this case the window manager is responsible for swapping colormaps when the pointer is moved in and out the x11 driver's window.

The default for 'mincolors' is maxcolors / (num_colormaps > 1 ? 2 : 8), where num_colormaps is the number of colormaps which are currently used by gnuplot (usually 1, if only one x11 window is open).

Some systems support multiple (different) visual classes together on one screen. On these systems it might be necessary to force gnuplot to use a specific visual class, e.g. the default visual might be 8bit PseudoColor but the screen would also support 24bit TrueColor which would be the preferred choice.

The information about an Xserver's capabilities can be obtained with the program 'xdpyinfo'. For the visual names below you can choose one of StaticGray, GrayScale, StaticColor, PseudoColor, TrueColor, DirectColor. If an Xserver supports a requested visual type at different depths, 'gnuplot' chooses the visual class with the highest depth (deepest). If the requested visual class matches the default visual and multiple classes of this type are supported, the default visual is preferred.

Example: on an 8bit PseudoColor visual you can force a private color map by specifying 'gnuplot*maxcolors: 240' and 'gnuplot*mincolors: 240'.

```
gnuplot*maxcolors:  <integer>
gnuplot*mincolors:  <integer>
gnuplot*visual:     <visual name>
```

, "

— X11 OTHER_RESOURCES —

By default the contents of the current plot window are exported to the X11 clipboard in response to X events in the window. Setting the resource 'gnuplot*exportselection' to 'off' or 'false' will disable this.

By default text rotation is done using a method that is fast, but can corrupt nearby colors depending on the background. If this is a problem, you can set the resource 'gnuplot.fastrotate' to 'off'

```
gnuplot*exportselection:  off
gnuplot*fastrotate:  on
gnuplot*ctrlq:  off
```

3.1.0.77 xlib

The 'xlib' terminal driver supports the X11 Windows System. It generates gnuplot_x11 commands, but sends them to the output file specified by 'set output '<filename>''. 'set term x11' is equivalent to 'set terminal xlib; set output "|gnuplot_x11 -noevents"'. 'xlib' takes the same set of options as 'x11'."

3.1.0.78 xlib

The 'xlib' terminal driver supports the X11 Windows System. It generates gnuplot_x11 commands, but sends them to the output file specified by 'set output '<filename>''. 'set term x11' is equivalent to 'set terminal xlib; set output "|gnuplot_x11 -noevents"'. 'xlib' takes the same set of options as 'x11'."

4 Graphical User Interfaces

Several graphical user interfaces have been written for ‘gnuplot‘ and one for win32 is included in this distribution. In addition, there is a Macintosh interface at ftp://ftp.ee.gatech.edu/pub/mac/gnuplot

Also several X11 interfaces exist. One of them is called xgfe. It uses the Qt library and can be found on http://www.flash.net/~dmishee/xgfe/xgfe.html

In addition three Tcl/Tk located at the usual Tcl/Tk repositories exist.

Bruce Ravel (ravel@phys.washington.edu) has written a new version of gnuplot-mode for GNU emacs and XEmacs. This version is based on the gnuplot.el file by Gershon Elber. While the gnuplot CVS repository has its own copy the most recent version of this package is available from http://feff.phys.washington.edu/~ravel/software/gnuplot-mode/

5 Bugs

Floating point exceptions (floating point number too large/small, divide by zero, etc.) may occasionally be generated by user defined functions. Some of the demos in particular may cause numbers to exceed the floating point range. Whether the system ignores such exceptions (in which case ‘gnuplot‘ labels the corresponding point as undefined) or aborts ‘gnuplot‘ depends on the compiler/runtime environment.

The bessel functions do not work for complex arguments.

The gamma function does not work for complex arguments.

As of ‘gnuplot‘ version 3.7, all development has been done using ANSI C. With current operating system, compiler, and library releases, the OS specific bugs documented in release 3.5, now relegated to ‘old_bugs‘, may no longer be relevant.

Bugs reported since the current release as well as older ones may be located via the official distribution site: http://www.gnuplot.info

Please e-mail any bugs to bug-gnuplot mailing list (see Section 1.3 [Seeking-assistance], page 4).

5.1 Old_bugs

There is a bug in the stdio library for old Sun operating systems (SunOS Sys4-3.2). The "%g" format for 'printf' sometimes incorrectly prints numbers (e.g., 200000.0 as "2"). Thus, tic mark labels may be incorrect on a Sun4 version of ‘gnuplot‘. A work-around is to rescale the data or use the ‘set format‘ command to change the tic mark format to "%7.0f" or some other appropriate format. This appears to have been fixed in SunOS 4.0.

Another bug: On a Sun3 under SunOS 4.0, and on Sun4's under Sys4-3.2 and SunOS 4.0, the 'sscanf' routine incorrectly parses "00 12" with the format "%f %f" and reads 0 and 0 instead of 0 and 12. This affects data input. If the data file contains x coordinates that are zero but are specified like '00', '000', etc, then you will read the wrong y values. Check any data files or upgrade the SunOS. It appears to have been fixed in SunOS 4.1.1.

Suns appear to overflow when calculating exp(-x) for large x, so ‘gnuplot‘ gets an undefined result. One work-around is to make a user-defined function like e(x) = x<-500 ? 0 : exp(x). This affects plots of Gaussians (exp(-x*x)) in particular, since x*x grows quite rapidly.

Microsoft C 5.1 has a nasty bug associated with the %g format for 'printf'. When any of the formats "%.2g", "%.1g", "%.0g", "%.g" are used, 'printf' will incorrectly print numbers in the range 1e-4 to 1e-1. Numbers that should be printed in the %e format are incorrectly printed in the %f format, with the wrong number of zeros after the decimal point. To work around this problem, use the %e or %f formats explicitly.

‘gnuplot‘, when compiled with Microsoft C, did not work correctly on two VGA displays that were tested. The CGA, EGA and VGA drivers should probably be rewritten to use the Microsoft C graphics library. ‘gnuplot‘ compiled with Borland C++ uses the Turbo C graphics drivers and does work correctly with VGA displays.

VAX/VMS 4.7 C compiler release 2.4 also has a poorly implemented %g format for 'printf'. The numbers are printed numerically correct, but may not be in the requested format. The K&R second edition says that for the %g format, %e is used if the exponent is less than -4 or greater than or equal to the precision. The VAX uses %e format if the exponent is less than -1. The VAX appears to take no notice of the precision when deciding whether to use %e or %f for numbers less than 1. To work around this problem, use the %e or %f formats explicitly. From the VAX C 2.4 release notes: e,E,f,F,g,G Result will always contain a decimal point. For g and G, trailing zeros will not be removed from the result.

VAX/VMS 5.2 C compiler release 3.0 has a slightly better implemented %g format than release 2.4, but not much. Trailing decimal points are now removed, but trailing zeros are still not removed from %g numbers in exponential format.

The two preceding problems are actually in the libraries rather than in the compilers. Thus the problems will occur whether ‘gnuplot‘ is built using either the DEC compiler or some other one (e.g. the latest gcc).

ULTRIX X11R3 has a bug that causes the X11 driver to display "every other" graph. The bug seems to be fixed in DEC’s release of X11R4 so newer releases of ULTRIX don’t seem to have the problem. Solutions for older sites include upgrading the X11 libraries (from DEC or direct from MIT) or defining ULTRIX_KLUDGE when compiling the x11.trm file. Note that the kludge is not an ideal fix, however.

The constant HUGE was incorrectly defined in the NeXT OS 2.0 operating system. HUGE should be set to 1e38 in plot.h. This error has been corrected in the 2.1 version of NeXT OS.

Some older models of HP plotters do not have a page eject command ’PG’. The current HPGL driver uses this command in HPGL_reset. This may need to be removed for these plotters. The current PCL5 driver uses HPGL/2 for text as well as graphics. This should be modified to use scalable PCL fonts.

On the Atari version, it is not possible to send output directly to the printer (using ‘/dev/lp‘ as output file), since CRs are added to LFs in binary output. As a work-around, write the output to a file and copy it to the printer afterwards using a shell command.

On AIX 4, the literal ’NaNq’ in a datafile causes the special internal value ’not-a-number’ to be stored, rather than setting an internal ’undefined’ flag. A workaround is to use ‘set datafile missing ’NaNq’‘.

Concept Index

E

F

G

H

I

K

L

U

V

W

X

Y

Z

Command Index

C

E

F

H

I

L

P

Q

R

S

T

U

Options Index

Function Index

Terminal Index

Table of Contents